Object Pose Estimation and Tracking with Deep Learning for Robot Manipulation

By

Tao Chen

A thesis submitted for the degree of

Doctor of Philosophy

School of Computer Science and Electronic Engineering

University of Essex

May 10, 2023

Abstract

Perceiving the 6D pose of object is a longstanding question. It plays a crucial role in some areas of robotics, such as object manipulation, grasping, unmanned aerial vehicles and autonomous vehicles. Although researchers have proposed various algorithms to address this problem in the history, like template matching, point pair feature, etc., object pose estimation in 3D space still remains challenging, especially at present as robots are being used more widely in complex environments. Currently existing algorithms cannot solve the problem well in terms of robustness and effectiveness. In this thesis, we utilise deep learning techniques to overcome the limitations of some existing pose estimation algorithms. Specifically, we investigate two different tasks in perceiving the orientation and translation of an object in 3D space, pose estimation from single images, and pose tracking from video sequences.

For the pose estimation task from single image, we introduce a novel channel-spatial attention network, which can learn the representative features from RGB-D images. Although there are some supervised Convolutional Neutral Network (CNN) frameworks used for estimating the object pose, they simply fuse the image features and geometry features together, which result in weak representations of fusion data as these multimodal data lays in various feature spaces. To address this, our channel-spatial attention network proposes a specific CNN that learns the most important embedding from each data format, and convert them to the identical dimensional feature space. Our experimental results show that the proposed framework can achieve better or comparable estimating result, even in the condition of clutter scenes, occlusions, and various illuminations. Furthermore, we propose a novel graph representation for the pose estimation task from single images in this thesis, in which not only the relationships between different sources of data, but the inter-connections of single data are exploited. Unlike the CNN which only captures local features, the proposed graph-based neural network considers the local and global features and demonstrates a more robust performance when some occlusions exist.

To extend the content of the pose estimation task from single image, in this thesis, the pose tracking task is also investigated. We propose a novel tracking framework that achieves stable and real-time tracking in the process. This framework is based on the correspondence of two consecutive frames, where the temporal-spatial information is utilised. A segmentation CNN is firstly used to locate the interest objects. Then the correspondences are established by a light-weight optical flow network. After that, the initial transformation can be predicted using 3D-2D and/or 2D-2D matching.

In summary, the goal of this thesis is to extend the research direction of pose estimation tasks in 3D space, especially by introducing some advanced deep learning techniques to this area. In this thesis, it shows that our deep learning based methods have the advantages of dealing with occlusions, and cutter background images over some existing object pose estimation methods.

Acknowledgement

I want to thank my supervisor, Professor Dongbing Gu, for his lots of advises to my study. His guidances and supports are important to finalize this thesis. Also, I want to thank my parents, for their support and encourage. I also want to thanks Dr. Mohan, Vishwanathan M for his help. Our robotics technician, Robin Dowling, Mark Marney, Jon Whitby, thanks their help for my experiments

Contents

List of Figures

List of Tables

Notation

$\mathbf{R}$	Rotation
$\mathbf{t}$	Translation
$\mathbf{T}$	Transformation Matrix
DOF	Degree of Freedom
$\mathbf{q}$	Quaternion
$\mathbf{SO}(3)$	Special Orthogonal Group
$\mathbf{SE}(3)$	Special Euclidean Group
PnP	Perspective-n-Point
ICP	Iterative Closest Point
RANSAC	Random Sample Consensus
SVD	Singular Value Decomposition
UAVs	Unmanned Aerial Vehicles
SVM	Support Vector Machine
CNN	Convolutional Neural Network
ROI	Region of Interest
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
ORB	Oriented FAST and Rotated BRIEF
MLP	Multilayer Perceptron
CAD	Computer-Aided Design
ADD	Average Distance of Model Point
VSD	Visible Surface Discrepancy
CSA6D	Channel-Spatial Attention Network
FPS	Farthest Point Sampling

AUC	Area Under Curve
$\otimes$	Element-Wise Multiplication
KNN	K Nearest Neighbour
PSPNet	Pyramid Scene Parsing Network
CBAM	Convolutional Block Attention Module
$\mathbf{F}_f$	Fusion Feature
$\mathbf{M}_s$	Spatial Attention Feature
$\mathbf{M}_c$	Channel Attention Feature
C	Confidence Score
$\mathbf{K}$	Camera Intrinsic Matrix
$\mathbf{G}$	Direct Graph
$\mathbf{V}$	Vertices
$\mathbf{E}$	Edges
h_θ	Non-Linear Function with Learnable Parameters θ
$ReLU$	Rectified Linear Unit
$\mathbf{F}_{Pixel}$	Pixel Feature
$\mathbf{F}_{Point}$	Point Feature
$\mathbf{F}_{Edge}$	Edge Feature
L_{pose}	Pose Regression Loss
L_f	Feature Loss
I_k	Observed Image at Time k
M_k	Mask of Object at Time k
$\mathbf{F}_{k-1,k}$	Pixel Motion from Time I_k to I_{k-1}
$\mathbf{F}_c$	Filtered Flow Consistency
w	Warping Operation
$\mathbf{F}_r$	Rigid Flow
$\mathbf{F}_d$	Difference between Filtered Flow Consistency and Rigid Flow
σ_{rigid}	Threshold for Rigid Flow Consistency Measure
$\Delta\mathbf{T}$	Pose Change
IMU	Inertial Measurement Unit

Chapter 1

Introduction

1.1 Motivations

Nowadays, we have seen booming in the use of different robots in a range of industrial fields. The object 6D pose estimation is the key element for them. For real world applications, the object 6D pose estimation algorithms must be reliable and precise. To achieve this goal, we need the auxiliary of algorithms and hardware components to build the system. For example, Tesla's self-driving system required 48 neural networks to train its vision system to recognize the cars and pedestrians, and/or other perception tasks. In total, its training took 70000 GPU hours [1]. In order that this giant framework be reliable, the neural networks should have the precision to handle complex situations on the road. Also, such training requires an intense of amount of data. Obviously, the powerful hardware can support the algorithm, allowing it to preform computation tasks. As is the case with virtual reality, we need to know the pose of an object to render it and its surrounding environment.

If 6D pose estimation techniques are applied to agriculture, they can be used to predict the pose of fruits to be grasped, and improve the performance and productivity of automated harvesting systems. In the scenarios of farm or city, we need to detect and localize the objects precisely in those unstructured environments. For example, to harvest the apples or strawberries planted on farms or orchards, the vision system is responsible for the detection and estimation of the fruits and provides the position and/or orientation information to the robot arm for further harvesting. As the last few years have revealed the powerful ability of learning based methods for dealing with image detection and segmentation, a popular

method of detecting fruits to be harvested is deep learning image segmentation techniques, the masks produced can establish the position of the fruits. But on some occasions, the 2D information is not sufficient to pick and/or cut the fruits without damaging them. We need extra information, such as the fruit's orientation, to guide the robot arm in the direction of the fruits to be harvested. Kim et al [2] proposed a deep convolutional network for identifying the pose of tomato and its side stems. It showed that estimating the pose of the tomatoes accurately can increase the rate of successful harvesting.

In addition to agriculture, 6D object pose estimation techniques can be applied to warehouse picking and place and bin-picking tasks. In these applications, 6D object pose estimation plays an important role in the perception stage, to help the robot to recognize the object and plan the motion of its gripper and/or manipulator. As in the warehouse, objects could be placed with random. 6D object pose estimation techniques are usually required to recover the object's pose without damaging it, or damaging other surrounding objects. Furthermore, the grasping pose is also quite important to a successful picking movement. To generate a suitable grasping pose, the object pose must first be estimated with respect to the end-effector. Based on the estimated object pose, some algorithms [3] can plan the graphing path to grasp objects stably. In the bin-picking task [4], the robot uses the 6D pose estimation technique to recognise the objects to be grasped and avoid the collision. Object pose information can help the house cleaning robot to locate the pose of an object, and guide the robot to manipulate it, for example opening the cap of water bottle, cleaning the table, etc.

Inspired by the motivations of object 6D pose estimation tasks, this thesis, aims to develop new object 6D pose estimation algorithms to accelerate their development. Our works are developed aim to accuracy and robustness of the estimation under some challenging situations, like occlusion, background clutter, and lighting variability.

1.2 Objectives

First of all, we define the specific in object 6D pose estimation. As shown in fig 1.1, a point $\mathbf{P}$ of object scissor is represented in coordinate frame T_o, in which we establish it as object coordinate normally. Suppose we can find the projection $\mathbf{P}'$ of $\mathbf{P}$ in the camera coordinate frame T_c. The point $(\mathbf{u}, \mathbf{v})$ is the camera principal point. The objective is to find the transformation ($\mathbf{R}$,$\mathbf{t}$) from T_o to T_c. To be able to find the transformation, the correspondences

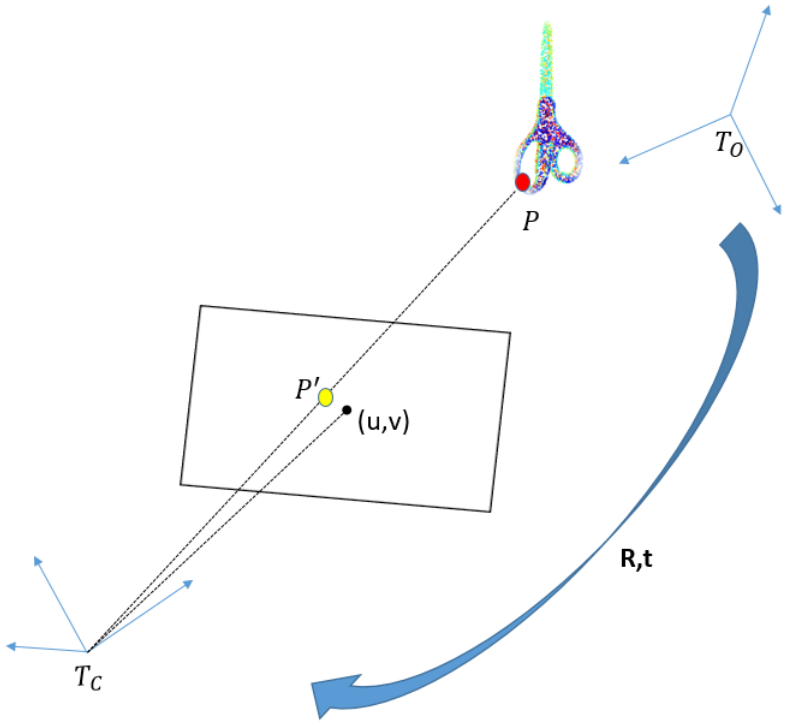

Figure 1.1: Illustration of object 6D pose estimation problem.

should be established. In the past few years, 6D object pose estimation has been investigated intensively in both research areas and industry applications. However, Finding such a transformation is not a trivial problem due to the reason that it is not easy to match them given a set of $\mathbf{P}$ and $\mathbf{P}'$. For example, the features of objects presented in fig 1.2 pose challenging to traditional methods.

Figure 1.2: Illustration of objects with occlusions, texture-less objects and symmetrical object. Images are taken from the T-LESS dataset.

This thesis focuses on estimating and tracking the object 6D pose using deep learning,

to overcome the limitations of traditional methods. Essentially, this thesis, develops the learning-based methods which can predict the 6D pose of an object from the input of an image. The fig 1.3 shows an end-to-end deep learning framework to estimate the 6D pose of an object presented in the image. This network takes image data for its input and directly outputs the 6D pose of an object. Generally speaking, the works of this thesis are to develop the aforementioned network.

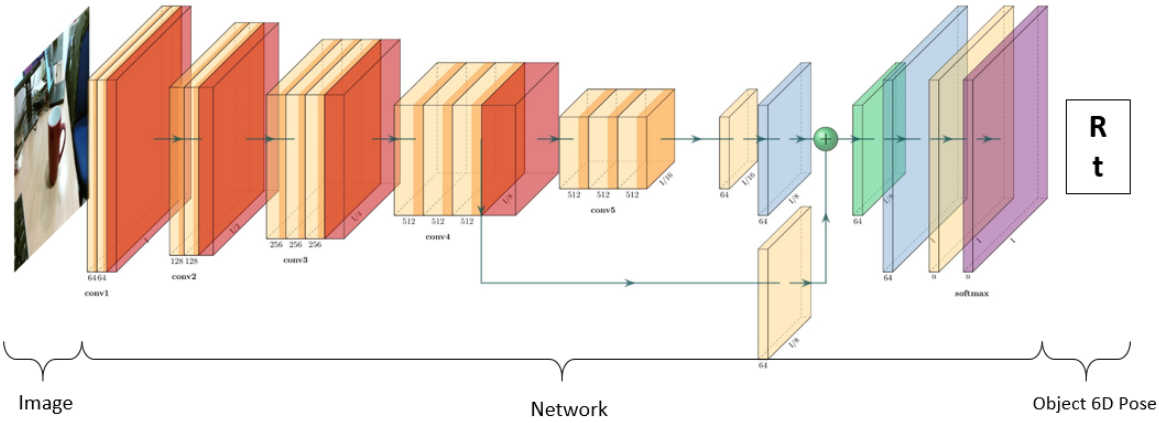

Figure 1.3: Illustration of an end-to-end network for object 6D pose estimation.

1.3 Contributions

One solution to object 6D pose estimation using RGB camera is to use 3D-2D correspondence. Suppose we know a set (more then 3 points) of 3D points of an object. We can then find its projection in image frame through a feature extraction algorithm (SIFT, SURF). In the end, the pose can be recovered by using PnP algorithms. In the past, this has been the popular method to deal with 6D object pose estimation problem. But this method is not stable when the objects to be estimated are texture-less, or being occluded. These scenarios also pose challenges scenarios when attempting to find the features of object. Also, when it is necessary to estimate multiple objects, the feature extraction process is time-consuming. Due to the significant progress of hardware in recent years, depth sensors and light detection and ranging (Lidar) sensors are used in variety of applications.

This also brings up more solutions for object pose estimation tasks. The geometry information provided by the use of a depth camera, can provide more accurate distance information than an RGB image when the object is projected on its frame. The increased used

of depth cameras can compensate for this information. Matching two point clouds, the pose can be obtained by solving a 3D-3D correspondence problem. Once the point clouds of the object in question are recovered from the data, we can use an off the shelf point cloud registration algorithm like ICP to obtain the optimized object pose. This kind of solutions usually needs the object model to be available. But the 3D-3D matching problem still remains, since traditional solutions significantly rely on calculating the specific local features, like the Point Pair Feature(PPF) for extracting feature from point cloud. But PPF is not robust when the point cloud is being occluded and/or reflective. Registration approaches are also challenging when the point cloud is very dense. In addition, it poses challenges when identifying the object of interest from a point cloud set.

To address the aforementioned limitations, deep learning is applied to the domain of 6D object pose estimation. We have seen the powerful ability of deep learning to solve the problems in computer vision and robotics research. The emergence of these learning approaches is beneficial to the progress of computational hardware and the availability of large-scale datasets. By using large amounts of data, machines can learn the knowledge representation of outcomes. So, by taking advantage of deep learning techniques, some researchers are trying to solve the pose estimation problems in regard of supervised and unsupervised learning. For example, in SLAM (Simultaneous localization and mapping) tasks, deep learning techniques are used to estimate the camera pose, depth map, and ego-motion. Object pose estimation works in contrast to camera pose estimation, which is concerned with predicting the pose of object in term of camera. Normally, the data for object pose estimation is obtained from the camera, which could be RGB images, RGB-D images, or depth images only. At the moment, RGB-D sensors are becoming popular in the field of robotics, as the price is affordable. Also, compared to RGB data or depth data, RGB-D data can provide both visual information and geometrical information, which provides a greater depth of information of the object under analysis. In fig 1.4, it shows a sequence of RGB image, depth image, and point cloud obtained by ZED sensor.

Among the learning-based approaches, CNN (Convolutional Neural Network) demonstrates some noticeable results. Even using holistic-learning RGB-D based methods, in which the pose is predicted in end to end manner, yielded remarkable results compared to traditional methods. Previous proposed frameworks for processing RGB-D data simply

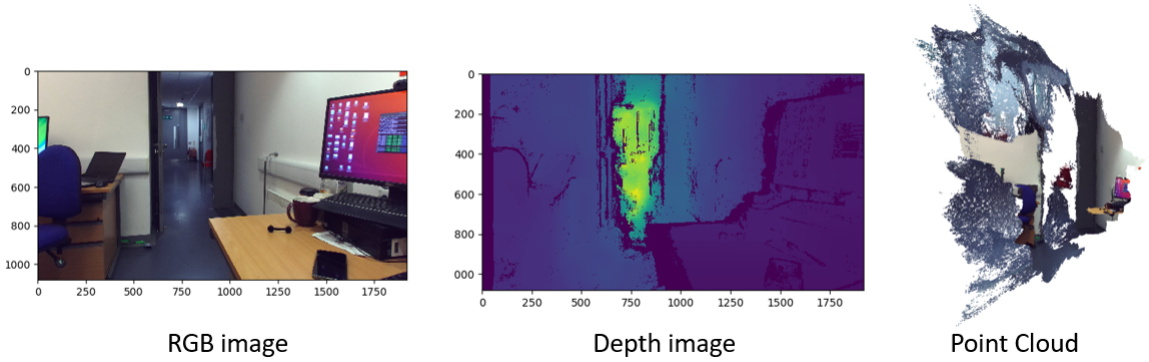

Figure 1.4: Illustration of RGB image, depth image and point cloud.

focused on extracting color information, and geometrical information, separately, and combined them together, like the works [5] [6]. This kind of fusion strategy could result in insufficient utilization of fusion features from a variety of different data sources. Thus, in chapter 3, it conducted the research by proposing an attention network to focus on extracting the representative features of channel and spatial dimension. First of all, the objects to be estimated will be localized by a segmentation network. Then, the segmented visual information and geometrical information are processed by the 2D features extraction network and 3D feature extraction network, respectively. Our attention network doesn't change the size of fusion features after the refinement. So, it is convenient to insert our attention module into the existing deep network. Once we obtain the refined fusion feature, the object 6D pose is regressed by multiple MLP branches, where one MLP branch is used for regressing rotation vector, one MLP branch for regressing translation vector, and one MLP branch for predicting a confidence vector. We tested our framework on several popular datasets used in object 6D pose estimation, and achieved remarkable performance, compared with selected state of the art methods. From the attention map generated by our network, we found that our network learns the features from the regions of object with sufficient features and avoids the occlusion parts. You can find our proposed architecture in fig. 3.1 of chapter 3.

We believe that there are two limitations the framework proposed in chapter 3,which prevent a better estimation performance, especially for the estimation of symmetrical objects. First, the network we used to extract the point feature treated each point in the point cloud independently, which ignores the local information. Secondly, directly regressing object pose by MLP layers from fusion features is challenging. The symmetry objects typically cause ambiguity during the learning process, as they have a similar feature representation

corresponding to different poses. So, in chapter 4, we utilized a graph network to learn the edge information between each point's neighbours. Also, we constructed our network in a way of Siamese architecture, in order to let them taking a pair of RGB-D samples for training. Furthermore, we designed a loss function that enforces the Euclidean distance to be minimized between the pose space and feature space. For testing, we only used one RGB-D sample as input. The result of experiment showed that training with our proposed loss function largely improves the pose estimation accuracy of symmetry objects.

In chapter 5, we extended the object 6D pose estimation to object 6D pose tracking for robotic manipulation. We utilised the optical flow network to predict the motion of the object. Based on the optical flow vectors we estimated, we proposed a keypoint selection scheme to select the corresponding keypoints from two consecutive input RGB-D frames. Also, to validate the selected keypoints, we used an iterative keypoint selection scheme to filter out the unreasonable keypoints. Then, the change in the object's pose between two frames is estimated by the 3D-2D correspondences of their keypoints. We evaluated our framework through the dataset for robotic manipulation, and demonstrated the robust and accurate pose tracking performance through the interaction of manipulator. Our network can still track the fast moving objects, with a high degree of accuracy, without losing the tracking. In order to indentify and track the category object, previous learning based methods needed to learn the features of its prior shape. Our method directly selects the keypoints based on motion information, and it can also achieve the tracking of the category-objects.

This thesis is organised as follows: chapter 1 clarifies our research motivations, objectives, and its contributions. Chapter 2 provides a literature reviews. In chapter 3, describes the details of my contribution for the object 6D pose estimation by an attention network. Chapter 4 demostrates the details of Siamese graph attention network for object 6D pose estimation. Chapter 5 describes the method of object 6D pose tracking by utilising optical flow estimation. Chapter 6 is the conclusion of this thesis.

1.4 Publications

Some of the contributions mentioned above have been submitted and published in internationally conferences and journals. Here is the list:

Chen, Tao, and Dongbing Gu. "6D Object Pose Estimation with Attention Networks."

2021 26th International Conference on Automation and Computing (ICAC). IEEE, 2021.

Chen, Tao, and Dongbing Gu. "CSA6D: Channel-Spatial Attention Networks for 6D Object Pose Estimation." Cognitive Computation 14.2 (2022): 702-713.

Chen, Tao, and Dongbing Gu. "6D Object Pose Tracking with Optical Flow Network for Robotic Manipulation", The 22nd World Congress of the International Federation of Automatic Control. Accepted.

Chapter 2

Literature Review

In the field of computer vision and robotics, object pose estimation has been well studied, especially with recent deep learning techniques. They demonstrate a powerful ability to deal with texture-less objects and background clutter where traditional methods cannot. This literature review discusses the most popular estimation methods,including the methods for 2D and 3D object estimation. The approaches discussed in this review are related to both traditional and convolutional neural network based estimation methods. We analyse several popular object detection and object pose estimation methods, such as 3D bounding box estimation, object coordinate regression, template-matching, and pose regression. In addition, the thesis reviews annotated 6D pose datasets for dealing with complex environments.

2.1 Preliminaries

In this section, we are going to review three commonly used pose representations for a rigid object, and two technical algorithms which we will use in our work to calculate the object pose. In the literature, to represent the movement of a rigid object, the common ways are the transformation matrix, quaternion and Lie group.

2.1.1 Representation of Object 6D pose

Generally, a transformation matrix $\mathbf{T}$ has the form:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix}, \mathbf{R} \in \mathbb{R}^{3\times3}, \mathbf{t} \in \mathbb{R}^{3\times1} \tag{2.1}$$

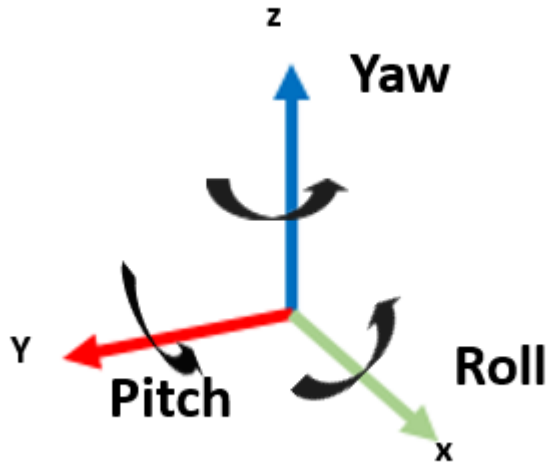

Figure 2.1: The rotation representation by Euler Angles.

where $\mathbf{R}$ indicates a rotation matrix and $\mathbf{t}$ a translation vector. Specifically, a rotation matrix has the properties, $\mathbf{R}\mathbf{R}^T = \mathbf{I}$, $det(\mathbf{R}) = 1$, where $\mathbf{R}^T$ is the transpose matrix of $\mathbf{R}$. The multiplication of a rotation matrix and its inverse is an identical matrix, and determinant of a rotation matrix is 1. Intuitively, we utilise three Euler angles decoupling the aforementioned R. In this way, a rotation can be expressed by singe rotation for x axis, y axis and z axis respectively, three times in total. Note that the sequence of rotation can start from any axis from x, y and z. In fig.2.1, we show the graphic expression for three Euler angles, Yaw(ψ), Pitch(θ) and Roll(ϕ). As we know, a general rotation matrix can be represented as a 3×3 matrix in the form:

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2.2}$$

The converted 2.2 according to Euler angle expression, it can be written as:

$$\mathbf{R} = R_z(\psi)R_y(\theta)R_x(\phi) \tag{2.3}$$

which means rotating the object along with roll angle, pitch angle and yaw angle respectively. Furthermore, equation 2.3 can be expressed as follows:

$$\mathbf{R} = R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta cos\psi + cos\phi sin\psi & cos\phi sin\theta cos\psi - sin\phi sin\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \tag{2.4}$$

While the Euler angles can represent the rotation more intuitively, they suffer from the problem called gimbal lock.

Quaternion. Due to the ambiguity of representing rotation using Euler angle, researchers are increasingly using quaternion to express rotation.

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{2.5}$$

It contains real part q_0, and vector part $q_1 i$, $q_2 j$ and $q_3 k$. i, j and k are the complex elements of the vector part. They satisfy the relationship $i^2 = j^2 = k^2 = ijk = -1$. Suppose a point $\mathbf{p} = [x, y, z] \in \mathbb{R}^3$ in the 3D space, rotates along with axis $\mathbf{n}$ for angle θ. The resulting point $\mathbf{p}'$ is obtained as:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1} \tag{2.6}$$

where $\mathbf{q}$ equals to $[cos(\theta/2), \mathbf{n}sin(\theta/2)]$.

Suppose a quaternion has a form as equation 2.5. From quaternion representation to rotation matrix, its corresponding rotation matrix is:

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \end{bmatrix} \tag{2.7}$$

Although quaternion requires fewer parameters than the representation for object pose when used in learning or optimization methods, it still requires the quaternion to be unit quaternion. Also, **q** and **-q** represent the identical rotation. In the case of camera pose estimation or object pose estimation, this constraint might increase the difficulty in obtaining the desired results. Thus, some researchers choose Lie group to represent the object pose in 3D space.

Lie group. The Lie group has the properties to optimise object or camera pose with no-constraints. As we know, a rotation matrix can construct a SO(3)(special orthogonal group) group.

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, det(\mathbf{R}) = 1\} \tag{2.8}$$

And a transformation matrix $\mathbf{T}$ can construct a SE(3) group(Special Euclidean Group).

$$SE(3) = \{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3\} \tag{2.9}$$

If we specify the rotation between -π and π, we can find the unique correspondence between Lie groud and Lie algebra. Suppose we have Lie algebra *se(3)*, $\rho \in \mathbb{R}^3$ representing a translation vector, and $\phi \in \mathbb{R}^3$ for a rotation vector in 3D space. We integrate them into the vector form $\zeta \in \mathbb{R}^6$, and its matrix form be

$$\zeta^\wedge = \begin{bmatrix} \phi^\wedge & \rho \\ 0 & 0 \end{bmatrix} \tag{2.10}$$

There exists an exponential mapping that can map the Lie algebra to the Lie group.

$$exp(\zeta^\wedge) = \begin{bmatrix} exp(\phi^\wedge) & J\rho \\ 0 & 1 \end{bmatrix} \tag{2.11}$$

where $exp(\phi^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!}(\phi^\wedge)^n$, $J = \frac{sin\theta}{\theta} + (1 - \frac{sin\theta}{\theta})aa^T + \frac{1-cos\theta}{\theta}a^\wedge$. θ and a are the magnitude and direction of vector ϕ, $\phi = \theta a$. The symbol $\wedge$ represents the transformation from the vector ϕ to its skew symmetric matrix $\phi^\wedge$

In reverse, we can also map a Lie group $\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in$ SE(3) to a Lie algebra representation $\zeta = \begin{bmatrix} \rho \\ \phi \end{bmatrix}$ with the logarithmic mapping. $\theta = arccos\frac{tr(R)-1}{2}$, $Ra = a$, $t = J\rho$. Solving the first two equations for θ and a, we can obtain ϕ as $\phi = \theta a$. Solving the third equation, we can find ρ

In the object 6D pose estimation domain, the aforementioned pose representations are used interchangeably.

2.1.2 Perspective-n-Point and Least Square Fitting

In our study, except for the learning methods that generate the object 6D pose straightforwardly, the PnP(perspective-n-point) and least square fitting methods are used to obtain the object 6D pose. These two methods have wide applications in the computer vision, and have been implemented in many open source tools, like OpenCV [7]. So, in this subsection, we are going to briefly introduce those two methods. We can also divide them into the categories: 3D-2D correspondence and 3D-3D correspondence. The 3D-2D estimation, used in this study is known as the PnP method, which utilises the geometry relation between the 3D points and their 2D projections in the image frame to refine the camera or object pose. 3D-3D correspondence is an optimization approach that finds the best transformation (R and T) for two point cloud sets, so that they can match with the minimal match error.

Perspective-n-Point.PnP can calculate the object pose when given a set of correspondences between 3D-2D point sets. A minimum 3 points in 3D space are required to get the object, and their projections in the image space. It should be noticed that the coordinates of points are presented in the world reference coordinate, not the camera coordinate. In most cases using PnP solution, we assume the camera is calibrated, which means the camera's intrinsic parameters are known. In addition, to reduce the impact of outliers in the input data, Random Sample Consensus(RANSAC) is commonly used together with PnP. In fig 2.2, we have three pairs of 3D-2D correspondences,$(A,a),(B,b),(C,c)$. O is the camera principal point. According to [8], we need the fourth point in order to address the ambiguity from the possible solutions of P3P. We denote this pair as (D,d) Let $v = \frac{AB^2}{OC^2}$, $uv = \frac{BC^2}{OC^2}$, $wv = \frac{AC^2}{OC^2}$, $x = \frac{OA}{OC}$, $Y = \frac{OB}{OC}$ we can construct two equations:

$$(1-v)y^2 - ux^2 - cos(b,c)y + 2uxcos(a,b) + 1 = 0 \tag{2.12}$$

$$(2-w)x^2 - wy^2 - cos(a,c)x + 2wxycos(a,b) + 1 = 0 \tag{2.13}$$

Wu-Ritt proposed a zero decomposition method to solve preceding equations [9]. However, P3P only utilises three points' information, therefore it cannot deal with the situation with more than 3 correspondences. So, EPnP [10] is proposed to address the problem of greater amount of correspondences.

Least Square Fitting. In the domain of camera pose estimation and object pose estima-

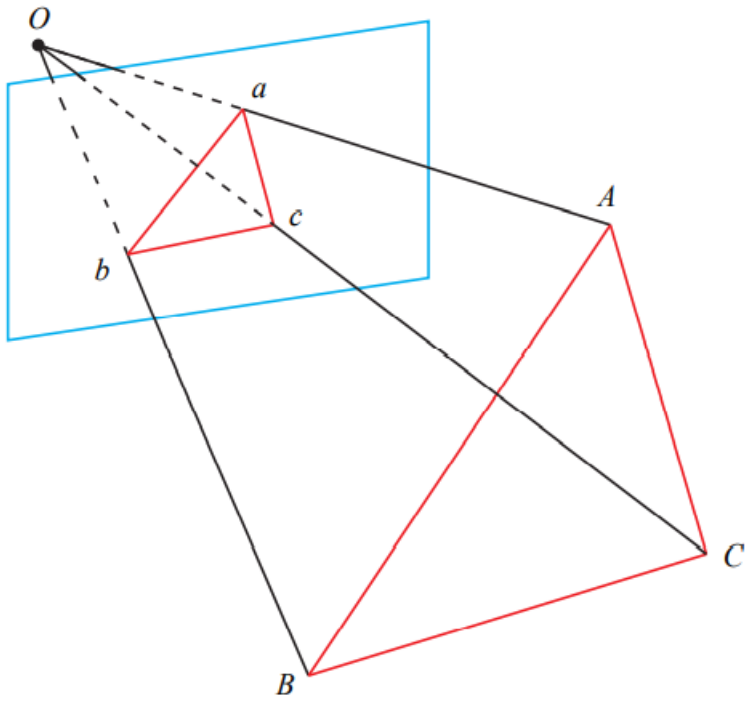

Figure 2.2: Illustration of P3P Problem

tion, we often encounter the problem of estimating the transformation between two matched point cloud sets. The ICP algorithm is a popular approach to address this kind of problem, and has been implemented in OpenCV and Open3D, and other open-source tools. Suppose we have two point sets with pre-established correspondences. $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \in \mathbb{R}^{3\times n}$, $\mathbf{P}' = \{\mathbf{p}'_1, \ldots, \mathbf{p}'_n\} \in \mathbb{R}^{3\times n}$, where each $\mathbf{p}$ and $\mathbf{p}'$ represents the 3D coordinates of point. Then, we want to find a $\mathbf{R} \in \mathbb{R}^{3\times 3}$, $\mathbf{t} \in \mathbb{R}^{3\times 1}$. Let $\forall i, \mathbf{p}_i = \mathbf{R}\mathbf{p}' + \mathbf{t}$. A general solution to obtaining $\mathbf{R}$ and $\mathbf{t}$ refers to works in [11], which is based on singular value decomposition(SVD). First of all, we define the error for the i-th point as:

$$e_i = \mathbf{p}_i - (\mathbf{R}\mathbf{p}'_i + \mathbf{t}) \tag{2.14}$$

Then we can construct a least square problem to find $\mathbf{R}$, $\mathbf{t}$ in order to make the error to be minimal.

$$\min_{\mathbf{R},\mathbf{t}} J = \frac{1}{2}\sum_{i=1}^{n} \left\|\mathbf{p}_i - (\mathbf{R}\mathbf{p}'_i + \mathbf{t})\right\|^2 \tag{2.15}$$

We set the centroids of two point cloud sets as $\mathbf{p} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}_i)$, $\mathbf{p}' = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}'_i)$. Hence, the equation

2.15 can be rewritten as:

$$\min_{\mathbf{R},\mathbf{t}} J = \frac{1}{2}\sum_{i=1}^{n} \left\|\mathbf{p}_i - \mathbf{p} - \mathbf{R}(\mathbf{p}'_i - \mathbf{p}')\right\|^2 + \left\|\mathbf{p} - \mathbf{R}\mathbf{p}' - \mathbf{t}\right\| \tag{2.16}$$

The term $\|\mathbf{p} - \mathbf{R}\mathbf{p}' - \mathbf{t}\|$, only relates to the object centriod. Once we can calculate the $\mathbf{R}$, the $\mathbf{t}$ can be easily obtained by setting this term to 0. Hence, the solution for ICP algorithm can be divided into three steps:

- Obtain the de-centralized coordinates for each point, $\mathbf{q}_i = \mathbf{p}_i - \mathbf{p}$, $\mathbf{q}'_i = \mathbf{p}'_i - \mathbf{p}'$
- Compute $\mathbf{R}$ according to $\mathbf{R}^* = arg\min_{\mathbf{R}} \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{q}_i - \mathbf{R}\mathbf{q}'_i\|^2$
- Calculate $\mathbf{t}$, $\mathbf{t} = \mathbf{p} - \mathbf{R}\mathbf{p}'$

For calculating $\mathbf{R}$, expand the equation:

$$\frac{1}{2}\sum_{i=1}^{n} \left\|\mathbf{q}_i - \mathbf{R}\mathbf{q}'_i\right\| = \frac{1}{2}\sum_{i=1}^{n} \mathbf{q}_i^T\mathbf{q}_i - 2\mathbf{q}_i^T\mathbf{R}\mathbf{q}'_i + \mathbf{q}_i'^T\mathbf{R}^T\mathbf{R}\mathbf{q}'_i \tag{2.17}$$

In the right side of equation 2.17, the first term is independent to $\mathbf{R}$, and the third term, as $\mathbf{R}\mathbf{R}^t = \mathbf{I}$. Hence, we can further simplify the optimization equation to:

$$\sum_{i=1}^{n} -\mathbf{q}_i^T\mathbf{R}\mathbf{q}'_i = \sum_{i=1}^{n} -tr(\mathbf{R}\mathbf{q}'_i\mathbf{q}_i^T) = -tr(\mathbf{R}\sum_{i=1}^{n}(\mathbf{q}'_i\mathbf{q}_i'^T) \tag{2.18}$$

We let $\mathbf{W} \in \mathbb{R}^{3\times3}$ as $\sum_{n}^{i=n} \mathbf{q}_i\mathbf{q}_i'^T$, then we apply SVD to $\mathbf{W}$:

$$\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^T \tag{2.19}$$

where Σ is a 3×3diagonal matrix with non-negative elements, $\mathbf{U}$ and $\mathbf{V}$ are real orthogonal matrices. When $rank(\mathbf{W}) = 3$, $\mathbf{R} = \mathbf{U}\mathbf{V}^T$. Then, according to $\mathbf{t} = \mathbf{p} - \mathbf{R}\mathbf{p}'$, we can get the $\mathbf{t}$. The details of provement can be found in the work of [11].

2.2 Introduction to 6D Object Pose Estimation and Tracking

There is a long history of the study of Pose estimation in field of computer vision.

Whether in the fields if industry applications, robotics manipulation or autonomous driving, it is crucial to understand the pose of an object froom the perspective of the camera or observers. In robotic applications, recovering object pose can help the robot interact with complex a environment. Pose estimation is particularly important in regard to manipulation tasks, such as the Amazon Picking Challenge(APC) [12]. The objects are grasped and manipulated from bin and shelf and placed elsewhere without any collision between them. If we know the pose of the object of interest, this task could be made faster and safer. The tasks of robot navigation, obstacle avoidance and path-planning also require the ability [13] [14] to guide the robot from one place to another. Pose information can also improve the performance of Unmanned Aerial Vehicles(UAVs) when GPS is not able to provide the service [15] [16].

The pose of object is defined by the relation between rotation and translation to the camera coordinate. In this review, most of the estimation methods use the method of supervised training in which object pose, 2D bounding box, and 3D model are provided. And the most challenging elements of object pose estimation are occlusion, clutter, similarities in appearance, texture-less and variability of dataset. The ingenious design of deep learning algorithms, has made some progress addressing aforementioned challenges, but still insufficient. Recent research to tackle occlusion and clutter problems, usually employs a voting scheme [17] [18] [19]. Unlike more traditional methods, which need to design specific feature descriptors for texture objects, the CNN-based model can be trained through massive annotated texture-less datasets to estimate the pose accurately. [20] [21].

In general, this review, will focus on the deep learning approaches that have emerged in recent years, in addition to some classical non-learning methods. In the development of deep learning, the convolutional neural network(CNN) shows the powerful ability for processing those visual images, especially in regard to image classification and image recognition. Most of the learning-based algorithms which estimate object 6D pose also rely on the advantages of those CNN detectors as the backbone network to extract features. This chapter has reviewed some popular CNN detectors. This thesis, also investigates object 6D tracking. Chapter 5 we proposes an algorithm to accomplish this. Essentially the methods mentioned in this chapter concern the object 6D pose estimation and tracking topics. Chapter 3 to 5 discuss out proposed algorithms in greater depth.

To summarise, the main method to recover object pose consists of four representations of object pose such as 3D bounding box, object-coordinate, template matching and directly regressing pose.

The remaining sections are organised as follows: subsection 2.3 briefly reviews several popular 2D image detectors which are used to get the 2D bounding box and segmentation of object of interest. Subsection 2.4 classifies the learning based and non-learning based object pose estimation approaches and subsection 2.5 reviews the works of the optical flow network. Subsection 2.6 reviews the works of object 6D pose tracking and subsection 2.7 explains the commonly used datasets for object pose estimation. Subsection 2.8 summarises the commonly used metrics for evaluating 6D pose estimation. Subsection 2.9 is the conclusion of chapter 2.

2.3 2D Detection and Segmentation

In the field of object pose estimation using CNN, a popular scheme was to segment target objects first, which was usually presented in the form of bounding box and segmented mask. In this way, we can remove the background and identify the object's location in the image, and move onto the next network for further processing. Using 2D detectors can extract the mid-level features of an object. These features can be used to regress object 6D pose, 3D bounding box, keypoints, for example. The following subsections will briefly discuss several 2D detectors used to estimate the object 2D bounding box and the semantic mask.

RCNN

Object detection posed a bigger problem than object classification. Not only did it require knowing the class of object, but also the bounding box which surrounds the object. The bounding box in the image is normally a rectangle, identified by its upper-left and bottom-right corners. Before CNN was adapted to object detection, the methods of sliding window and selective search [22] were widely used to propose bounding box. These methods were time-consuming and ineffective. In [23], the author introduced a region proposal method followed by a convolutional neural network called R-CNN. Firstly, about two thousand random proposals were produced through a selective search, and then each proposal was extracted to a fixed size as input to the convolutional network, and finally a support vector machine(SVM)

was used to classify each proposal.

Fast RCNN

Although R-CNN used CNN in detection, it still has several drawbacks. For instance, R-CNN is not an end-to-end network, it cannot update all layers at one time. Furthermore, from the record of [24] it took 47s in VGG16 per image in detection. Fast R-CNN was an end-to-end framework which was used to improve the detection result compared to R-CNN. In contrast to R-CNN, Fast R-CNN takes whole images as input and uses them to produce its feature map. In order to deal with arbitrary shape region of interest, the author introduced a layer called RoI pooling that extracted a fixed-length vector as input for a remaining fully connected layer. There were two branches at the end of architecture, one of which estimates the softmax probability over K object classes and plus one background, the other the four numbers of each class which represent object bounding box coordinates. The RoI pooling layer played the key role in achieving end-to-end training. For example, we specified the feature region we want to extract as $H \times W$ (they were 7×7 in the original paper), and then the corresponding shape of region of interest was $h \times w$. The RoI pooling layer downsized the RoI $h \times w$ into $H \times W$ by dividing it to approximately $h/H \times w/W$ sub-windows and then applying maxpooling to each window. In this way, we can obtain a fixed size feature-vector before sending it to the next FC layers.

Faster RCNN

Both R-CNN and Fast R-CNN can be seen as a kind of region proposal method. Fast R-CNN inputs whole images to perform convolution, and convolutional features will be shared with each region proposal. Faster R-CNN [25] was proposed to achieve real-time detection performance. Faster R-CNN comprised of two modules, the Region Proposal Network(RPN) for producing region proposals, and Fast R-CNN.

Generally, RPN directed the Fast R-CNN network to find the most likely region to belong to the object of interest in the feature map. Once the input image has been operated by shared convolutional layers, RPN will insert the last convolutional feature map, for example, taking $n \times n$ region on feature map as input. It mentioned that there were 9 different scales of region proposals in which it has a scale correspondence to original image. In [25], each position has 9 anchors on it and RPN will determine whether each anchor contained objects or not

and outputed the approximated coordinates of bounding box.

For the detection part, the author used the work of Fast R-CNN in order to train two networks jointly to share convolutional computation, rather than training them, independently. The alternating training was the first to be adopted by the RPN. Generated region proposals were then used to train Fast R-CNN, and then the results of Fast R-CNN will be used to initialize RPN.

Mask RCNN

Mask R-CNN [26] was a tiny changed network compared with Faster R-CNN. It added one more branch that identified the semantic segmentation of each object. That meant that each pixel in each image has to be classified coorespondingly. Mask R-CNN's network architecture is identical to RPN in first the stage, except it added one mask branch in parallel to object classification and bounding box regression branch. Also, it was different from the RoI pooling layer in Fast R-CNN, and it proposed a RoIAlign layer that reduced ambiguity at feature extraction.

As in the aforementioned, the RoI pooling layer will approximately divide the region of interest into $[h/H] \times [w/W]$ bins, where $[\cdot]$ meant rounding operation. This quantization has little impact on classification and small translation, but it has large negative impact on pixel-wise segmentation. To get accurate pixel level segmentation results, RoIAlign didn't use any quantization, it used the result of h/H instead. The bilinear interpolation was to compute the nearby 4 points of h/H, and then perform maxpooling operations on interpolation results.

Fully Connected Neural Network

Beside the region proposal method, a fully connected neural network(FCN) [27] was also wildly used to segment objects. The FCN replaced the last fully connected layer in CNN with a fully convolutional layer such that it can produce the probabilities for each pixel in the image rather than a fixed vector which represented the classification of the object. At the end of FCN, the map will generate an input image of the same size. To get its prediction, upsampling, some time called deconvolution, transpose convolution in other literatures, was used to make sampling feature maps the same size as input image. To obtain more accurate semantic segmentation results, the previous pooling layers can be unsampled and summation with final layer unsample result.

Except the aforementiioned network architectures, ResNet [28] was also a popular network for learning object detection. To address the degradation of deep neural network, ResNet proposed an identity short-cut connection that skipped one or more layers, which forced the network to learn the residual information, especially for the deeper layers. With this simple connection, it didn't increase the parameters of the network and computational complexity. Also, ResNet can construct very deep architecture, ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-150 and other more deep variants, where number followed by ResNet indicates the layers it has. DenseNet [29] also achieved the good performance for feature learning. It considered the short-cut connections which connected each layer to all the other later layers, to allow the maximum flow of information to go through the network. Encoder-Decoder architecture [30] was also a popular choice for semantics segmentation and was used to learn the representation of the object in object 6D pose estimation.

2.4 Object 6D Pose Estimation

In this section, the works of object pose estimation were reviewed. Subsection 2.5.1 and subsection 2.5.2, reviewed some popular estimation approaches using traditional feature detection and matching methods. From subsection 2.5.3 to subsection 2.5.9, we reviewed the methods of deep learning for object pose estimation. In section 2.6, the methods for optical flow estimation were discussed, as optical flow estimation was a crucial part of the framework proposed in chapter 5. Section 2.7 reviewed the object 6D pose tracking methods,

2.4.1 Traditional Methods for Object 6D Pose Estimation

In this subsection, we reviewed some popular and effective traditional methods in our research domain. The term "traditional" was compared to those learning methods. It didn't mean that they were useless in today's researches and applications.

In fig 2.3, we showed a classical method of estimating object 6D pose. Suppose a RGB image was taken as the input for this pipeline. The feature detection block was used to detect the keypoints on the image. For instance, the ORB feature was a powerful feature to describe the image. Alternatively, computer vision community also adopted SIFT and SURT. As there were some features that belonged to background, we needed the auxiliary of object model to match the features of the object of interest. In the PnP+Ransac block, we can determine

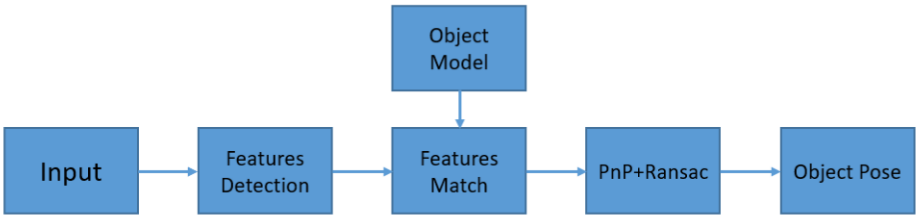

Figure 2.3: Object 6D pose estimation pipeline with feature detection and feature matching fashion.

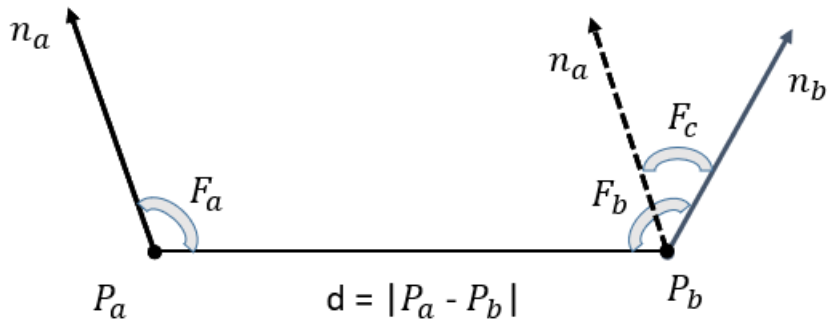

Figure 2.4: Illustration of point pair feature.

the pose of the object by the 3D-2D correspondences established in the previous block. Ransac was used to remove the outliers from the estimation. But this kind of framework has limitations, in that it requires that the object to be detected has enough texture information. In industry applications, many objects are texture-less. Also, this pipeline requires object model which makes the framework more applicable to specific objects.

Point Pair Feature. [31] first proposed this kind of approach to detect the object pose from point cloud data. Unlike the image feature detectoo, which described the local information of pixels, point pair feature utilised the geometrical relationship between two points to match scene point clouds to object model. In fig 2.4, we used two points to show how to construct the point pair feature. In there, we have two points, $\mathbf{P}_a$ and $\mathbf{P}_b$. Their Euclidean distance $\mathbf{d}$ was one of the element in point pair feature. Then, it computed the normals of two points, $\mathbf{n}_a$ and $\mathbf{n}_b$, then angle with normal and the distance vector $\mathbf{d}$ were taken as the elements of point pair feature, noted as $\mathbf{F}_a$ and $\mathbf{F}_b$ in the figure. The last element was the angle formed by two normal vectors, $\mathbf{F}_c$. So, one point pair feature can be expressed as:

$$\mathbf{P}(\mathbf{p}_a, \mathbf{p}_b) = (||\mathbf{d}||, \angle(\mathbf{n}_a, \mathbf{d}), \angle(\mathbf{n}_b, \mathbf{d}), \angle(\mathbf{n}_a, \mathbf{n}_b)) \tag{2.20}$$

To calculate the point pair feature, we needed the object 3D model. The points in object model will be perturbed and calculate point pair feature for each point pair. So, in this way, we have a global representation of the object, unlike the local description as provided by the image detector. The resulting features of object model will be stored in a hash table. Point pair features were also caculated for the scene point cloud(input data). To get the object pose, each computed feature of object model was matched with computed features of scene, and a Hough-voting scheme was used to vote for the highest possibility of object pose for the object. The limitation of using point pair feature was that there would exist erroneously detection of simple object, especially planar objects, as there would be many similar features.

2.4.2 Template-Based Methods

The most traditional template-based method probably was [32]. It was proposed by Hinterstoisser et al. In template-matching method, object model was required to provid multiple viewpoints. Multiple templates will be sampled in relation to the object. [33] it stated 2000 samples could yield a more accurate result. We then extract color gradient and surface normal from the input image using the sliding window and compare them with the templates that were stored in memory. The authors also proposed a dataset called *LineMod* which was commonly used in the object pose estimation. But the condition of lighting remained consistent in this dataset.

2.4.3 3D Bounding Box Methods

3D bounding box was a cubic that covered object tightly and fixed the its pose properly. It was described as a tuple (x,y,z,w,h,l, θ_r, θ_p, θ_y) where (x,y,x) was the bottom corner of 3D bounding box, (w,h,l) was width, height and length of 3D bounding box and (θ_r,θ_p,θ_y) was the roll, pitch, and yaw angle, respectively. For those scenes where objects had to be placed on the ground, as in the case of the car, the method would only consider yaw angle [34]

[35] predicted 2D projection of corners of 3D bounding box and calculated pose using Perspective-n-Point(PnP) algorithm. For the training, it only required the ground-truth of 3D bounding box. Rad et al. [36] used the segmentation results to reasoning 2D projection of 3D bounding box instead of using 2D bounding box. In [37], depth image was added to combining with RGB image for regressing 3D bounding box. It used a CNN to extract visual features from RGB input and a PointNet [38] to extract geometrical information after

input pre-processing. Then resulted features were then used to directly predict box corners. To deal with occlusion, Oberweger et al. [39] predicted heatmaps of corners of 3D bounding box projection on image plane from multiple image patches.

2.4.4 Pose Regression-Based Methods

Inspired by the success of Mask R-CNN, Toan Do et al. [27] proposed LieNet, which directly regressed object pose from only RGB image as input. The LieNet used a VGG-16 network as its feature extractor and shared four head branches which were used for bounding box regression, object classification, mask segmentation and 6D pose regression. The pose regression branch was a multilayer perceptron that produced rotation matrices in the representation of Lie algebra. LieNet achieved the state-of-the-art 6D object pose estimation result while using only an RGB image, without post-refinement. With addition to LieNet, PoseCNN [5] has been proposed by directly regressing object pose. PoseCNN used VGG16 network architecture as its feature extractor, and followed three embedding steps that produced semantic labels, translation vectors and rotation vectors, respectively. Unlike other methods that directly regressed the object centre is cases of object translation, PoseCNN added a Hough voting layer where the result of semantic labels and result of center regression were embedded as input into the Hough voting layer. The locations with maximum voting score were selected as object centers. The advantage of the leveraging voting method was to make the model robust in the clutter scene by utilising visible pixels. Within contrast to LieNet, PoseCNN applied the quaternion format. Although the representation of quaternion avoided the ambiguities inherent in the representation of Euler angle, quaternion has limitation that its unit norm must be one which imposed the limitation of value range of network output. Also PoseCNN needed ICP algorithm for post processing.

When combined with a single RGB input, depth image can compensate for depth information. The work of DenseFusion [6] leveraged these two different types of data to regress object pose. DenseFusion demonstrated an effective CNN-based method that leveraged these two sets of complementary data can into one the state-of-the-art object pose estimation result. The DenseFusion detector combined embededing features that came from segmented RGB images and geometrcal features produced by PointNet [38]. The fusion was performed in per-pixel level in which object appearance information and geometrical information were

explicitly combined together, showing the potential to overcome occlusion and the clutter scene. Also, the detector was followed by a refinement network to improve estimation accuracy. Inspired by the fusion between distance and color information, Gao et al. [40] argued geometrical information obtained from point clouds can only infer pose information. The point cloud used as input passed to two estimation networks which were similar to PointNet but removing the spatial transformer blocks. They adopted the axis-angle representation for the rotation regression.

2.4.5 Pose from RGB Image

There were many deep learning architectures that performed excellently in object detection [41] [25] and segmentation tasks [26]. Normally, 2D bounding box and segment mask of object were cropped from the image, then formed the input for CNN-based 6D pose estimation approaches. Kehl et al [34] first used a single-shot SSD detector to find the interested objects, then their viewpoints were estimated through classification instead of regressing 6D pose numerically. But this method was inaccurate when applied to real world examples. Park et al [42] used an auto-encoder to make pixel-wise predictions for an object's 3D coordinate, obtaining the pose by solving 2D-3D correspondence by using PnP algorithm. They also attempted to use a Generative Adversarial Network [43] to determine the occlusion part of the object, but their work had complex training stages, which could pose a certain level of difficulty for real world deployment. To consider each pixel's contribution, Peng et al [18] made use of a voting scheme to select the best keypoint from each pixel's prediction. Similarly, Xiang et al [5] proposed a network that regressed the center of object as translation from pixels and regressed their distance to the centre directly. Due to the discontinuity of each object's rotation space, it was hard to predict numbers as rotation vectors directly from a single RGB image. Briefly most RGB methods used a 2D detector as their backbone for feature extraction, and its effectiveness in estimating 6D pose from extracted features has been demonstrated.

2.4.6 Pose from RGB-D/Point Cloud Data

RGB-D/Point cloud methods: To understand 3D scenarios, PointNet pioneered the classification and segmentation of 3D data. To overcome the limitation of PointNet that treated individual points in 3D space independently, VoxelNet [44] encoded 3D points into regular

voxels. Although voxelization can locally exploit the geometry information of point cloud, it will result in plenty of empty voxels which were not memory and computation friendly. Gao et al [40] utilised two PointNet-like networks to direfctly regress object pose from un-ordered point sets. Wang et al [6] proposed an iterative fusion network that took RGB-D image as input. In their work, the pose was estimated from the combined features of RGB information and depth information. In 3D detection tasks, Sindagi et al [45] proposed a similar multi-modality fusion strategy for vehicle detection using the KITTI dataset. In their work, they obtained image feature map from a pre-trained Faster-RCNN network, while point cloud was projected back to image plane using camera intrinsic information. The point features were appended with corresponding image features, then a Voxel Feature Encoding layer which was obtained from VoxelNet, used to encode aggregated information.

2.4.7 Object Coordinate Regression

Within contrast to the approaches that directly regressed pose information, object coordinate regression methods, which regressed 3D object coordinates, established the 2D-3D correspondence that was used to calculate pose by PnP and RANSAC [10] [21]. This method often required object model, and can be divided into RGB-only and RGB-D input methods. In [42], the authors proposed a pipeline called Pix2Pose that used a single RGB input patch to estimate object coordinates. The input patch was obtained from 2D object segmentation containing the object of interest. The network was an auto-encoder that predicted coordinates, so ground-truth coordinates were calculated by rendering object models with ground-truth pose.

2.4.8 Attention Mechanism

The model with attention mechanisms have shown a remarkbale performance in the take of machine translation. The Transformer [46] almost became a standard network in the domain of natural language processing. Not only in the sequence-based (text, audio) tasks, but also in image and 3D data processing. It was a powerful technique to enhance the feature representation learned from those data. Woo et al [47] proposed that, processing a given feature map, in terms of different dimensions can focus its attention map on necessary information. For graph-structure data, Veličković et al [48] introduced a network architecture with an attention layer which considered the contributions from different neighbors of a node. Fur-

thermore, attention mechanism has a vast array of potential applications in video responses to questions, video captioning, and video recognition.

2.4.9 Transformer-based Object 6D Pose estimation

Transformer [46] demonstrated its superiority for processing text data in the domain of natural language processing and its multi-head self-attention mechanism has been widely incorporated into the deep neural network for processing image data. The work of [49] built a vision transformer that spilt the images into a number of patches, and then they were processed by a pure transformer. Its experimental results proved that it could achieve the same performance as CNN in image classification tasks. Recently, transformer-based networks also demonstrated the competitive performance on the object 6D pose estimation task. Amini et al. [50] proposed an architecture for pose regression of multiple objects. It used ResNet50 [28] for the extraction of visual features, and positional encoding information to enter into the transformer encoder-decoder, followed by three layers MLPs to regress object pose. As the same author of [50], they proposed an advanced network called YOLO-Pose [51], which worked as predicting the 2D projections of 3D keypoints in the image. Thereafter, the object pose can be recovered by PnP algorithm. 6-ViT [52] was a transformer-based instance representation learning framework, which can be used to estimate object 6D pose. It was constructed in a similar way to the network we proposed in Chapter 3. It took two various modalities as input, RGB and point cloud, respectively. To process RGB patch, it created the *Pixelformer* module to extract appearance embeddings, which has a transformer-based encoder-decoder architecture. To process point cloud source, it proposed *Pointformer* which has a cascaded transformer encoder and an all-MLPs decoder to extract geometrical information. Then, a multi-source aggregation network was used to unify the representation of fused embeddings. By adding the shape-prior information to the fused representation, 6-ViT can also predict the 6D pose for category-level objects. In the end, two MLP branches were used to reconstruct the instance point representation of the object, where the object pose can be recovered from observed point cloud and reconstructed point cloud, using a least-square estimation.

2.5 Optical Flow Estimation

Optical flow describes the pixels' motion in the consecutive frames, and the amount of elapsed time. In general, the optical flow algorithm makes several assumptions. Here we list two important assumptions.

1 The pixel's intensities remain constant during two consecutive frame.

2 Neighbouring pixels have similar motion.

We can summarise these two assumptions in one equation.

$$\mathbf{I}(\mathbf{x},\mathbf{y},t) = \mathbf{I}(\mathbf{x}+dx,\mathbf{y}+dy,t+dt) \tag{2.21}$$

Applying the taylor series approximation, finally we can get the equations.

$$\mathbf{f}_x\mathbf{u}+\mathbf{f}_y\mathbf{v}+\mathbf{f}_t = 0 \tag{2.22}$$

where

$$\mathbf{f}_x = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}, \mathbf{f}_y = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}, \mathbf{u} = \frac{\partial \mathbf{dx}}{\partial dt}, \mathbf{v} = \frac{\partial \mathbf{dy}}{\partial dt} \tag{2.23}$$

$\mathbf{f}_x$ and $\mathbf{f}_y$ are the image gradients. $\mathbf{u}$ and $\mathbf{v}$ are the motion in $\mathbf{x}$ and $\mathbf{y}$ direction along time dt, respectively.

There are two kinds of optical flow normally used in applications, sparse optical and dense optical. Spare optical flow tracks the specific features of pixel motion, such as corners. Hence, it requires using some dedicated algorithms(SIFT,SURF) to pre-process the images before calculating the sparse optical flow. In contrast, dense optical flow calculates the motion of entire pixels in the image.

In the scenario of object pose estimation and tracking, the objects to be estimated could be complicated, especially illumination variation. These kind of situations make it very challenging to predict optical flow when using traditional algorithms. But recently, deep learning techniques have been used to predict the optical flow, and show the high accuracy and robustness.

This section reviews the learning-based optical flow estimation algorithms, as optical flow estimation played a key role in the work of Chapter 5, where optical flow was used

to find the correspondence between two frames. FlowNet [53] was the first proposed CNN network to predict the optical flow. FlowNet consists of two variants, FlowNetSimple and FlowNetCorr. FlowNetSimple only utilized convolutional operation to process input data, where two input frames were simply stack together. Instead of using the generic convolutional network, FlowNetCorr uses two sepatrate but identical networks to process the input frames and concatenate them together in the later stage. To force the network to learn the meaningful correspondence bwtween two images, a correlation layer is proposed to multiply the feature maps before they are stacked together. Training a supervised optical flow network requires a large labelled optical flow dataset, which is unlikely to label each pixel's motion by human operator. The authrs of FlowNet, also published a synthesis dataset, **FlyingChair**, to help the training of optical flow network. FlowNet2 [54] is an advanced version in comparison to FlowNet, more equipped to deal with pixels with small displacements, and more robust to process real-world data. For the large displacement case, FlowNet2 uses multiple FlowNetSimple networks to learn it, in which the second image is warped with the predicted flow, then passed to the next stage. For small displacement, it replaces the stride operation in FlowSimpleNet with a smaller stride operation. Finally, the predicted results from large displacement and small displacement are fed into a fusion network to obtain the final flow prediction. In the case of PWC-Net [55], it establishes a pyramid network to process the input images. Specially, it uses up-sampled flow generated from a higher level pyramid to warp the first image, and then proceeds to a cost volume layer, which stores the mismatched costs of corresponding pixels, Then using cost volume as input, the final flow is estimated using a multi-layer CNN. Since optical flow estimation has a wider range of applications, it needs to be more friendly to resource-limited devices, like running on embedded systems. Hence, the light-weight optical flow networks [56], [57] are designed for the trade-off between speed and computation budget.

Below, we showed some qualitative results obtained from SimpleFlowNet and LiteFlowNet on NOCS dataset and YCB-Video datase, respectively.

Fig. 2.5 and Fig 2.6 are visualizations of optical flow estimated by SimpleFlowNet on NOCS dataset and YCB-Video dataset, respectively. Fig 2.7 is the visualization of optical flow of the LiteFlowNet prediction. The color indicates the motion direction, and its intensity is the displacement of the motion.

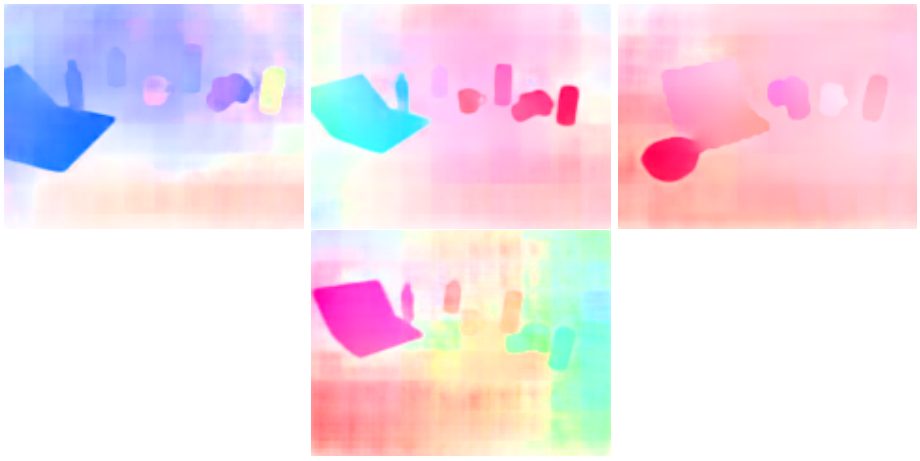

Figure 2.5: Examples of optical flow estimation on NOCS dataset (small camera movement) by SimpleFlowNet.

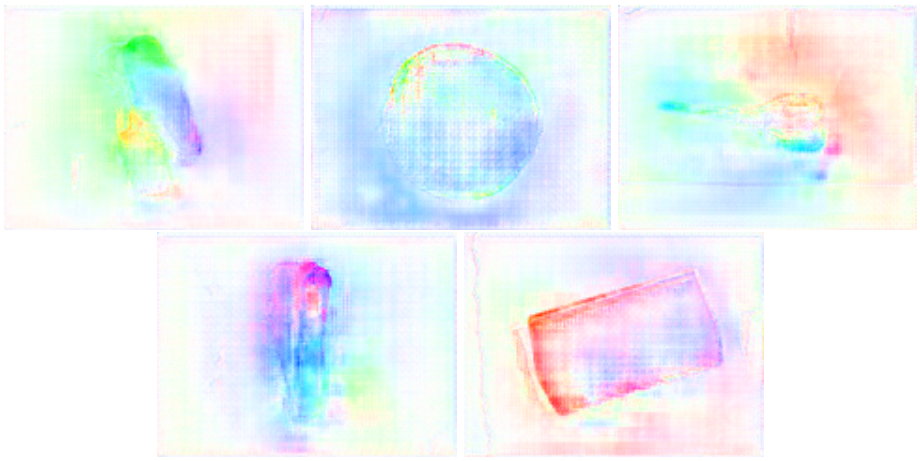

Figure 2.6: Examples of optical flow estimation on YCB-Video by SimpleFlowNet.

2.6 Object 6D Pose Tracking

In this section, we are going to review the recent application of object 6D pose tracking methods. As with above subsections, we focus on reviewing the works related to deep learning methods, but some traditional tracking algorithms will also be covered. Generally, learning based object pose tracking and object pose estimation have some similarities, for example, most of object pose estimation networks can be used in object pose tracking to predict the object pose from static frame. There are several factors that must be considered when using object pose tracking methods. In tracking, we usually have the observation before getting the pose result. Most of the time, we need to consider the spatial-temporal relation to determine the current pose from the previous and current observations. The classical Bayesian filtering

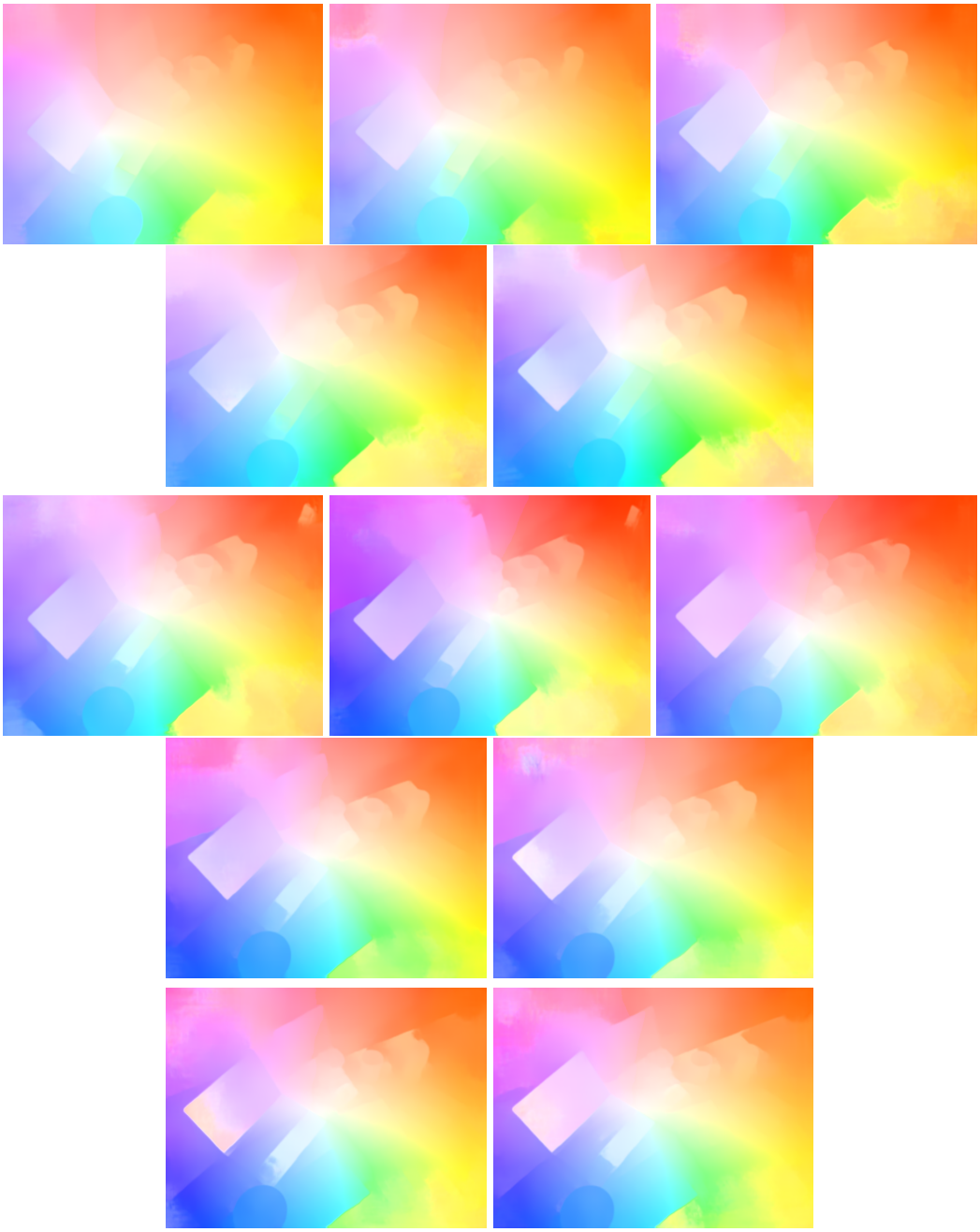

Figure 2.7: Examples of optical flow estimation on NOCS dataset by LiteFlowNet.

methods can be used to do this, which we will review in the next subsection. We can also use contemporary recurrent neural network, graph neural network and CNN to deal with this situation. As in tracking, an object might encounter occlusion during its movement, therefore the tracking algorithm must be robust to address this issue in order to avoid losing of the object being tracked. In robotics applications, the object pose tracking algorithms might also be interrupted by other objects or manipulators.

2.6.1 Object 6D Pose Tracking by Optimization

To overcome the challenges faced by the objects, such as self-occlusion, in-distinguish view, some works have tried to estimate the probability distribution of object pose. This estimation method models the uncertainty during the process of prediction. Here we are going to review some optimization methods for predicting 6-DOF object pose without the aid of deep learning. Karman filter and Particle filter are a popular and flexible tracking algorithm for tracking 2D or 3D objects, and they are commonly used to fuse the motion prior and the measurement information. A simple particle filter tracking pipeline could be as follows: Given the initial 6-DOF object pose distribution and initial weights, the sampled particles are propagated to predict the pose in the next frame using the motion model. Next, we must re-calculate the weights of particles using likelihood function. Then, the updated weights are used to re-sample the particles. The updated pose can be obtained by calculating the mean of re-sampled particles. Like in this fashion, Choi et al. [58] proposed a particle filter based tracking algorithm with high efficiency GPU implementation. They take point coordinates, point colors, and normals as the measurements for the likelihood function. Also, it needs to render the object model with predicted pose to the calculation, which makes it limited for some applications. In the work of [59], they utilised a Siamese network to predict the object 6-DOF pose with the quaternion representation. Combining particle filter and deep neural network, [60] utilised Rao-blackwellized particle filters to decouple the rotation and translation. First of all, they create a codebook by training an auto-encoder on photorealistic images, which are generated by discreting the 3D orientation space into the fixed number bins. So, it maps each rotation vector to a latent variable, which will later be used as observation likelihood functions. Then, the sampled particles with translation distribution are propagated to crop the predicted Regions of Interest(ROIs) where the object of interest could be included. The cropped regions will be sent to an auto-encoder to predict their corresponding latent variables, and compare them with the pre-computed codebook to update the distribution of 3D object orientation.

2.6.2 Object 6D Pose Tracking by Multi-Views

Some methods leverage images taken from various view points to realize the tracking process. In these methods, these multi-view images often have a spatial-temporal relationship

because they are usually obtained from a time sequence. In this subsection, the methods reviewed mainly involve taking a pair of images as input, which are normally the previous observation and the current observation. Generally, these images can be real observations from the camera during the tracking process. These cases normally rely on a segmentation network to crop the patches that encapsulate the objects of interest. Feature matching is then performed by a deep neural network or other classical feature detectors. On the other hand, some methods use only the current observation as reference image and take the auxiliary of object CAD model rendered by previous pose estimate, which is intended to provide camera intrinsics. Hence, the residuals between real observation and rendered image can be measured through different criteria, such as measuring the differences between the contours of two objects or the displacement of keypoints.

As in the work [61], it utilises the feature matching and registration by a deep CNN to calculate the relative pose change of an object. Two consecutive RGB-D images are given as input, first using a segmentation network to determine the region of interest. Then, a deep CNN is used to find the keypoint correspondence between two segmented images. The depth point cloud can be recovered from the 2D keypoints and corresponding depth pixels. Then, the relative pose change of object can be estimated from two depth point clouds by ICP algorithm. The work of $se(3)$-TrackNet [62], renders the object model as a reference image for the previous estimated pose, and takes the current observation image to predict the feature residuals between them. Then, based on the feature discrepancies, two separate branches are used to estimate the relative transformation via Lie Algebra, $\xi = (t, w) \in se(3)$. Marougkas et al [63] proposed a multi-attentional framework which takes an observed image and a rendered image to track a single known object. They generated two attention maps; one is the attention weight for foreground, another is the attention weight for occlusion of object. In the work of [64], Manhardt et al. propose an object 6D pose refinement network, which can also be used in the context of tracking. To update the pose change, they use the object CAD model and a pose hypothesis to generate a synthesis scene of object without background. Thereafter, they crop the patches from rendered scene and real scene, followed by the deep neural networks to extract low-level and high-level features. Then, the pose change is regressed from those extracted features.

2.6.3 Others Object 6D Pose Tracking Methods

There are some other methods that we cannot review comprehensively. For example, some works train their deep network purely on the synthetics data [65] [66], to avoid the human-interaction to label the pose information.

2.7 Datasets

This section, demonstrates several popular datasets used for posing estimation and/or tracking. As we know, data is crucial for any learning-based algorithms, especially for 6D object pose estimation. Obtaining pose information require an accurate and stable sensor configuration. Researchers have to consider how to annotate pose dataset more quickly and accurately. At the moment, we have several datasets which are widely used in the research of pose estimation. They perform different roles like pin-picking applications, grasping applications and others related to the robotics or computer vision field. The objects in those datasets focus on different properties, such as texture-less objects, symmetry objects, occluded objects and objects with clutter background. Normally, there are two stages of the dataset, the training set and test set with annotated ground-truth information. The object models will also be creating manually or obtained through 3D reconstruction techniques like KinectFusion [67], in the case of some frameworks require it.

- **LineMod and LineMod-Occluded**: The LineMod dataset is provided by Hinterstoisser et al [68]. 15 texture-less objects are placed on the table in the cluttered background. The dataset contains 15 image sequences, but each image sequence only has been annotated by one object. The lighting condition is constant in the dataset, with no significant occlusion with any of the other objects. Based on the LineMod dataset, Brachmann et al [33] provide an advanced dataset called LineMod-Occluded. In this advanced dataset, the authors add all the annotations for each test image sequence, and various lighting conditions, and object occlusion. That makes thg new dataset more challenging.

- **YCB-Video** : The YCB-Video dataset is provided along with the PoseCNN detector [5]. The dataset has 21 household obejcts from 92 videos with 133,827 frames. Apart

from real captured images, there are around 80,000 synthetic images provided. Also the object CAD models have been given.

- **T-LESS** : T-LESS [20] is a popular dataset at the moment used for the estimation of texture-less objects or symmetry objects. It is designed for the estimation of texture-less objects in regard to potential industry-relevant applications. Objects in T-Less are challenging because they have no texture, and all objects have a similar color and shape. The dataset is obtained through three time-synchronised different sensors (Microsoft Kinect v2, Canon IXUS 950 IS, Primesense Carmine 1.09). The training set has 38,000 images from each sensor with a plain black background, while the test set has 10,000 images from each sensor with a more complex background. The two different types of object models are also provided, one is the manually created CAD model, another is the semi-automatically produced.

- **MVTec ITODD** : Contrary to the aforementioned dataset, MVTec ITODD [21] focuses on industry applications. The dataset contains 28 rigid objects with different shapes and surfaces from 800 scenes, captured by two industrial stereo cameras and three gray-scale cameras.

- **HB(HomebrewedDB)**: The HB [69] dataset is the dataset designed to deal with texture and texture-less objects, scalability, occlusion and various light condition. Note that the training data in HB dataset is rendered by the 3D reconstructed models instead of real data. Using HB dataset, [17] [70] demonstrated that training on synthetic data can also achieve better generalization compared with using real data.

The datasets mentioned above are commonly used in the pose estimation research community. To achieve robust performance, however, data augmentation might be needed to make dataset generalizing well.

2.8 Metrics for Evaluating the Object 6D Pose

To analyse the accuracy of the object pose produced by the algorithms, we need the criterion to compare the predicted value with ground-truth value. It assesses the performance of the network, and can give us an intuitive measurement of the results. In this subsection, we

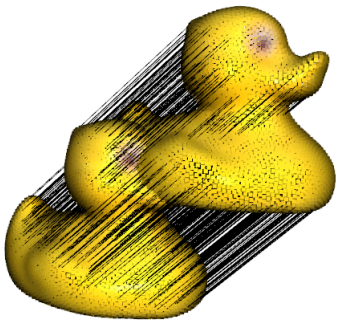

Figure 2.8: Illustration of distance calculation of ADD metric for non-symmetrical object.

review several commonly used metrics in the field of object 6D pose estimation, some of which will be used from Chapter 3 to Chapter 5.

Average Distance(ADD) The average distance is proposed in [33] in the first time. It is most wildly used in the evaluation for object 6D pose estimation and it has two variants(ADD, ADD-S) which aim for non-symmetrical and symmetrical objects, respectively.

$$ADD = \frac{1}{m} \sum_{x \in M} \left\| (\mathbf{R}x + \mathbf{t}) - (\tilde{\mathbf{R}}x + \tilde{\mathbf{t}}) \right\| \tag{2.24}$$

In formulation of ADD, $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{t}}$ are the predicted rotation and translation, while $\mathbf{R}$ and $\mathbf{t}$ are the ground-Truth. x denotes the points selected from the 3D model of object of interest. The predictions of rotation and translation are considered correct if the score of ADD is under a predefined threshold. As we can see in fig 2.8, the object model is transformed by the predicted pose and ground-truth pose. The black lines indicate the distance of corresponding points between two transformed point clouds. Obviously, the distance will be more small if the predicted pose is more close to ground-truth pose.

This threshold is normally considered as a constant value that describes coarseness of estimation multiply the diameter of the 3D model. Note that this metric cannot deal well with the objects with symmetric shape. Like illustrated in fig 2.9, calculating like ADD metric for symmetrical object will result in huge penalties for the rotation that are equivalent to symmetrical axis. Therefore, to evaluate the model's performance on symmetric objects, the ADD-S is used. In the formulation of ADD-S, the score is calculated as average distance to the closest point, and x_1 and x_2 are selected from the same 3D model. In fig 2.10, it shows

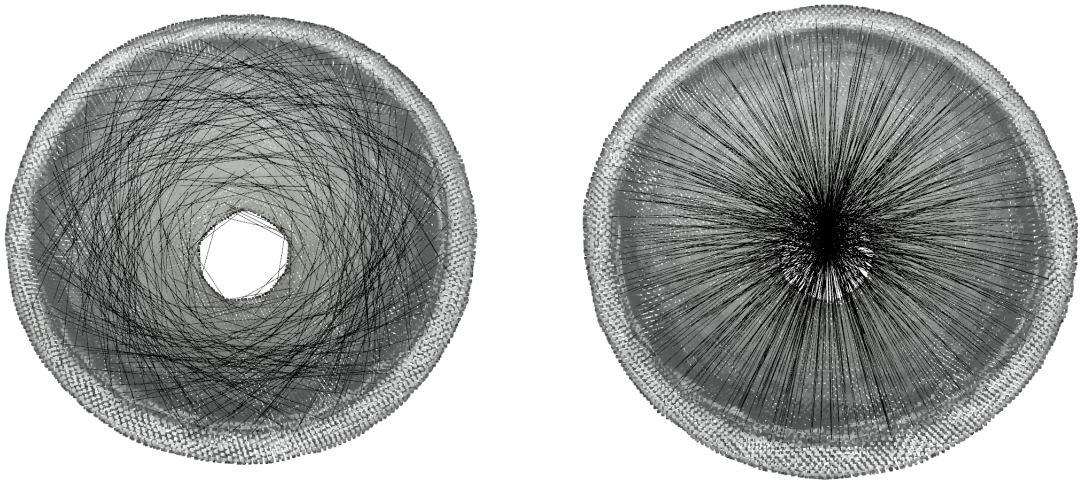

Figure 2.9: Illustration of distance calculation of ADD metric for symmetrical object. It shows two transformations that have different rotations(90 degrees and 180 degrees) but with identical translations.

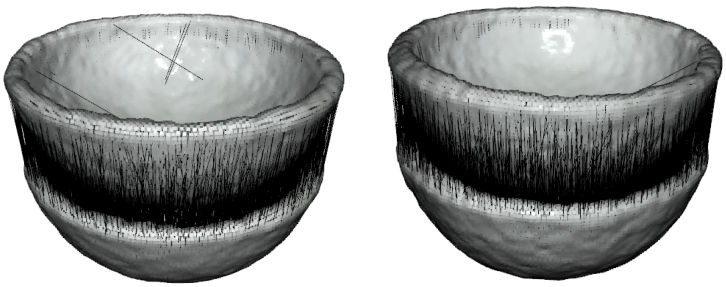

Figure 2.10: Illustration of distance calculation of ADD-S metric for symmetrical object.

two different rotations(90 degrees and 180 degrees) and same translations. The objects in the bottom are transformed by their ground-truth pose. We can see that the corresponding between predicted point cloud and ground-truth point cloud are established by the their nearest points where their average distances are close to each other even rotating with different degrees along with z-axis

$$ADD-S = \frac{1}{m}\sum_{x_1 \in M} \min_{x_2 \in M} \left\| (\mathbf{R}x_1 + \mathbf{t}) - (\tilde{\mathbf{R}}x_2 + \tilde{\mathbf{t}}) \right\| \tag{2.25}$$

Visible Surface Discrepancy(VSD) The method VSD requires depth image to calculate the visible mask of the object of interest. If we know the ground-truth pose and estimated pose, we can render the distance map $\mathbf{D}$ and $\tilde{\mathbf{D}}$, respectively, and these two generated images

are intersected with the depth image of test image to obtain visibility masks $\mathbf{V}$ and $\tilde{\mathbf{V}}$. The VSD can determine whether the pose estimation is correct or not by comparing two generated visibility masks.

$$e_{VSD} = avg_{p\in\tilde{\mathbf{V}}\cup\mathbf{V}} \begin{cases} 0, & p\in\tilde{\mathbf{V}}\cap\mathbf{V}\wedge|\tilde{\mathbf{D}}(p)-\mathbf{D}(p)|<\tau \\ 1, & otherwise \end{cases} \tag{2.26}$$

Where $|\tilde{\mathbf{D}}(p)-\mathbf{D}(p)|$ is the distance between the surfaces of two rendering models.

2D Projection This method computes the average distance from 2D projection. It takes 3D object models as input and then projects the model's vertices onto an image plane based on ground-truth pose and estimated pose. The AD is measured by calculating the distance between two projected points on the image if the average re-projection error e_{2Dproj} is less than 5 pixels. The formulation can be written as below:

$$e_{2Dproj} = \frac{1}{|\mathbf{V}|}\sum_{v\in\mathbf{V}}||\mathbf{K}\tilde{\mathbf{R}}v-\mathbf{K}\mathbf{R}v||_2 \tag{2.27}$$

where corresponding pixels are computed by coordinates of vertices $\mathbf{v}$ left multiply object pose R and then left multiply camera intrinsic matrices $\mathbf{K}$. The point cloud of object of interest can also be used as its 3D model.

Rotation and Translation Error To directly measure the numerical error between the predicted pose and ground-truth pose, we can use the performance metric proposed in [71]. Suppose we have predicted pose $\hat{\mathbf{P}} = \{\hat{\mathbf{R}},\hat{\mathbf{t}}|\hat{\mathbf{R}}\in SO(3),\hat{\mathbf{t}}\in\mathbb{R}^3\}$ and ground-truth object pose $\mathbf{P} = \{\mathbf{R},\mathbf{t}|R\in SO(3),\mathbf{t}\in\mathbb{R}^3\}$. The rotation error can be calculated as:

$$r_{err}(\hat{\mathbf{R}},\mathbf{R}) = arccos(\frac{Tr(\hat{\mathbf{R}}^T\mathbf{R})-1}{2}) \tag{2.28}$$

where Tr indicates the trace of the matrix. The translation error can be calculated by the L2 norm of two translation vectors.

$$t_{err} = ||\hat{\mathbf{t}}-\mathbf{t}||_2 \tag{2.29}$$

Comparison In order to comprehensively evaluate the performance of various methods on 6D object pose estimation dataset, this section provides statistical records of different methods using average distance as an evaluation metrics.

Method	input	ape	bench vise	cam	can	cat	driller	duck	box	glue	Hole punch	iron	lamp	phone	MEAN
Brach et al. [72]	RGB-D	98.1	99.0	99.7	99.7	99.1	100.0	96.2	99.7	99.0	98.0	99.9	99.5	99.6	99.0
Kehl et al. [73]	RGB-D	96.9	94.1	97.7	95.2	97.4	96.2	97.3	99.9	78.6	96.8	98.7	96.2	92.8	95.2
Wang et al. [6]	RGB-D	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
Hinterstoisser et al. [32]	RGB-D	95.8	98.7	97.5	95.4	99.3	93.6	95.9	99.8	91.8	95.9	97.5	97.7	93.3	96.6
Tekin et al. [70]	RGB	21.6	81.8	36.6	68.8	41.8	63.5	27.2	69.6	80.0	42.6	75.0	71.1	47.7	56.0
BB8 [36]	RGB	40.4	91.8	55.7	64.1	62.6	74.4	44.3	57.8	41.2	67.2	84.7	76.5	54.0	62.7
LieNet [74]	RGB	38.8	71.2	52.5	86.1	66.2	82.3	32.5	79.4	63.7	56.4	65.1	89.4	65.0	65.2
Pix2Pose [42]	RGB	58.1	91.0	60.9	84.4	65.0	76.3	43.8	96.8	79.4	74.8	83.4	82.0	45.0	72.4
SSD-6D [34]	RGB	-	-	-	-	-	-	-	-	-	-	-	-	-	76.3
DPOD [17]	Synthetic/RGB	87.7	98.4	96.0	99.7	94.7	98.8	86.2	99.9	96.8	86.8	100.0	96.8	94.6	95.1

Table 2.1: The performances of some state of art algorithms for object 6D pose estimation on LineMod dataset.

TABLE I compares the performances of different methods using RGB or RGB-D as input evaluated on LineMod dataset. The table shows that using RGB-D as input can achieve a better performance. Note that the DPOD method [17] takes synthetic data in training process while using real data for testing.

2.9 Conclusion

In this chapter, we focus on reviewing the methods that utilise convolutional networks to estimate object 6D pose. Due to the success of 2D image detection by CNN, objects of interest can be localised accurately, such as in the representation of its bounding box and segmentation. A resized segmented image patch can be used as input to regress object pose directly. If an object's model is provided, the detector can predict a 2D projection of a 3D bounding box and 2D-3D correspondences can retrieve object pose using PnP algorithm. Since the 2D appearance is projected from the real object, some efforts tried to predict object coordinates in the world-coordinate system. Then coarse pose is given by PnP algorithm and RANSAC removes outliers. Template-matching extracts color-gradient and surface normal of the object of interest as template searching on input image's space.

Conveying 6D object pose estimation to pose tracking, many existing frameworks for object detection and object pose estimation can be utilised to implement the object pose tracking framework. Optimization approaches combine the motion priors and current measurement to predict the pose distribution of the object of interest. We can also utilise the CNNs to find the features between two temporal-related images, and recover the object pose from their correspondences. The estimate of optical flow provides a straightforward way to determine information regarding the motion of the object. Thanks to the well-designed architectures and the advances in hardware, those CNN-based optical flow networks can predict optical flow quickly and accurately.

A dataset of high quality is essential to training a good performance network. The model is usually tested on LineMod, YCB-Video, T-LESS datasets where 6D pose annotations are available, but the pose estimation datasets are not limited to the aforementioned. To evaluate the performance of the 6D pose detector, we need the metrics to evaluate the gap between predicted pose and the ground-truth pose. Hence, we review some commonly used metrics in this chapter.

Chapter 3

Channel-Spatial Attention Networks for 6D Object Pose Estimation

6D object pose estimation plays a crucial role in robotic manipulation and grasping tasks. The aim to estimate the 6D object pose from RGB or RGB-D images is to detect objects and estimate their orientations and translations relative to the given canonical models. RGB-D cameras provide two sensory modalities: RGB and depth images, which could benefit the estimation accuracy. But the exploitation of two different modality sources remains a challenging issue. In this chapter, inspired by recent works on attention networks that could focus on important regions and ignore unnecessary information, we propose a novel network: Channel-Spatial Attention Network (CSA6D) to estimate the 6D object pose from RGB-D camera. The proposed CSA6D includes a pre-trained 2D network to segment the interested objects from RGB image. Then it uses two separate networks to extract appearance and geometrical features from RGB and depth images for each segmented object. Two feature vectors for each pixel are stacked together as a fusion vector which is refined by an attention module to generate a aggregated feature vector. The attention module includes a channel attention block and a spatial attention block which can effectively leverage the concatenated embeddings into accurate 6D pose prediction on known objects. We evaluate proposed network on two benchmark datasets YCB-Video dataset and LineMod dataset and the results show it can outperform previous state-of-the-art methods under ADD and ADD-S metrics. Also, the attention map demonstrates our proposed network searches for the unique geometry information as the most likely features for pose estimation. From experiments, we

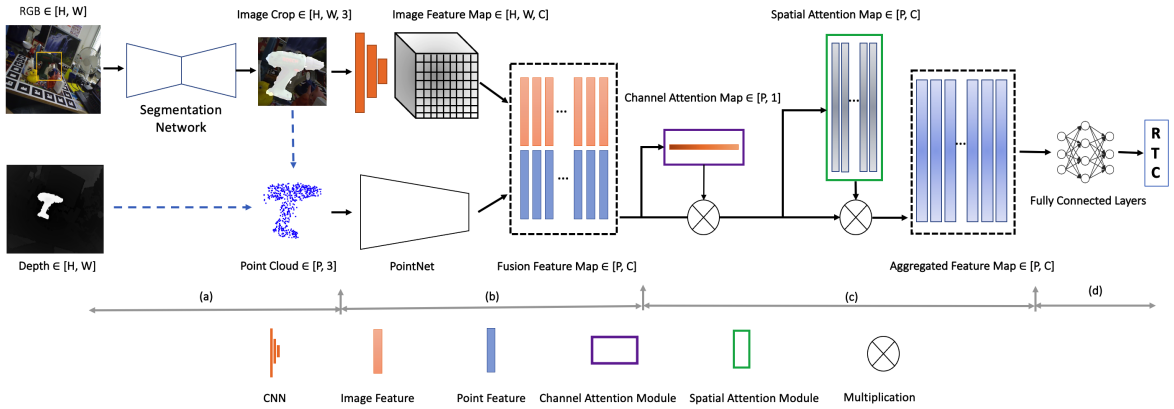

Figure 3.1: Overview of our proposed framework: (a) image segmentation and point cloud re-construction, (b) appearance and geometrical feature extraction, and feature fusion, (c) attention module for feature refinement, and (d) pose regression. The inputs are RGB image and depth image. The final outputs are **R**:rotation, **t**:translation, C:confidence.

conclude that the proposed network can accurately estimate the object pose by effectively leveraging multi-modality features.

3.1 Introduction

The aim to solve 6D object pose estimation problem with RGB or RGB-D images is to detect objects and estimate their orientations and translations relative to the given canonical models. It is a long standing problem in computer vision and robotics communities. Potentially the solutions to the problem could be applied to robot manipulation [75–77], self-driving cars [78, 79] or augmented reality [80, 81]. There are still some challenging issues in solving the problem when the images include severe occlusions, cluttered background, lighting variations, texture-less objects, or symmetrical objects.

Traditionally geometrical methods were used to solve the problem by matching RGB image features with object's 3D models [33,68]. These methods require well-designed hand-crafted features which are not robust to lighting variations, background clutters, or texture-less objects.

Recently deep learning methods have been proposed to solve the problem as CNNs have shown significant robustness to environment variations. Some of them took a holistic method to train end-to-end neural networks and regress the 6D pose directly from the networks [5]. Some of them exploited a key-point method and solved the problem with two stages: first estimate 2D keypoints of the object using deep networks and then estimate the 6D pose via

2D-3D correspondences with a PnP algorithm [82, 83]. Dense methods were also explored where a feature is extracted for each object pixel or patch and then the best minimal set of points is selected via the RANSAC algorithm or each feature casts a vote for 6D pose hypotheses [84].

RGB-D cameras have made two data modalities (RGB images and depth images) easily available and further pushed the research front for better 6D pose estimation. Some of RGB-D methods first estimated an initial pose from RGB image and then refined it on point clouds using an ICP algorithm or other optimisation algorithms [85, 86]. Others used two separate networks for RGB images and 3D point cloud to extract appearance and point-wise geometrical features, then concatenated both features to regress the 6D pose [6].

Recently attention mechanisms have shown a remarkable performance in deep learning applications. The Transformer [46] becomes very effective in natural language processing. Not only in the sequence-based (text, audio) tasks, but in image and 3D data tasks it is a powerful technique to enhance the feature representation learned from those data [87–90]. Woo et al [47] proposed an attention module that can process a given feature map in terms of spatial and channel dimensions to focus on necessary information. For graph-structure data, Veličković et al [48] introduced a network architecture with an attention layer which considers the contribution from different neighbors of a node. Like the aforementioned attention mechanisms, our attention module is also a self-attention method, but we do not generate weighted value for each feature, we refine the feature directly using Channel attention module and Spatial attention module, and it's simple than Transformer's multi-head self-attention.

In this chapter, we propose a novel end-to-end network: Channel-Spatial Attention Network (CSA6D) for 6D object pose estimation from RGB-D images. The proposed CSA6D includes a pre-trained 2D network to segment the interested objects from RGB image. Then it uses a 2D image detector and a 3D point cloud detector to extract appearance features and point-wise geometrical features from each segmented object. Two feature vectors for each pixel are stacked together as a fusion vector. Next it uses an attention module to process the fusion vector along spatial and channel axes to obtain an aggregated feature vector. Finally the 6D object pose is directly estimated from the aggregated feature vector via fully connected layers.

Our innovation is the use of an attention module to refine the fusion feature vector alone spatial and channel axes to improve the representation of feature map, and this design leads to a considerable accuracy improvement, so post-processing step is unnecessary in our model. In previous work [6], the fusion feature vector is directly fed to stacked MLP layers to regress the output. Here we argue that this process might not exploit the potential of all the information well while our proposed attention module could focus on more important features for pose regression. Since two modality features are simply blending together, the spatial-attention and channel-attention blocks are used to extract related representative features from their embedding space while keeping the original structure. Specifically, this design makes a robust representation for the modality fusion scheme and does not require a costly refinement step.

We evaluate our model on the LindMod dataset [68] and YCB-Video dataset [5]. The quantitative result shows that our proposed model can achieve a result with the state of the art accuracy compared with other learning methods.

3.2 CSA6D architecture

The architecture of our CSA6D is depicted in Fig. 3.1. An input RGB-D image with 640 x 480 pixels is fed to the system. Firstly, the RGB image is segmented by using a semantic segmentation network and each interested object is cropped from the image with its corresponding 2D bounding box and masks. By finding the corresponding region in the depth image with object masks, the object 3D point cloud is recovered by the camera calibration matrix and cropped depth region. The pre-trained segmentation network we used is an encoder-decoder network Mask R-CNN [26]. This segmentation network outputs $N+1$ binary maps in which each pixel belonging to that class (background class included) is activated and N is the number of object classes. The image patch that contains interested object is cropped by using 2D bounding box obtained from the segmentation network.

Secondly, we extract appearance features of the object from the cropped image patch using a CNN. Here we use Pyramid Scene Parsing Network (PSPNet) [91] as the appearance feature extractor to obtain an image feature map shown as the top branch in Fig. 3.1. The resulted feature map has size $H \times W \times C$ where C represents the dimension of each pixel in their feature space. H and W are the height and width of the original image patch. We extract

geometry features from the 3D point cloud data using a variant of PointNet [38] shown as the bottom branch in Fig. 3.1. The correspondence between two features for each pixel is established by using projection.

Thirdly, as appearance features in RGB image and geometry features in depth are complementary, they are stacked together as a fusion vector to form a compact representation of the interested object. We apply an channel attention block followed by a spatial attention block to refine the fusion feature. More specifically, the attention blocks perform max-pool and average-pool operations in channel and spatial axes to get a new aggregated feature vector that has same dimensionality with the fusion feature vector.

Finally we have three separate branches to estimate the rotation, translation and confidence, respectively, each of them using five fully connected layers. The confidence score refers to the confidence the network has on each prediction.

3.2.1 Attention Module

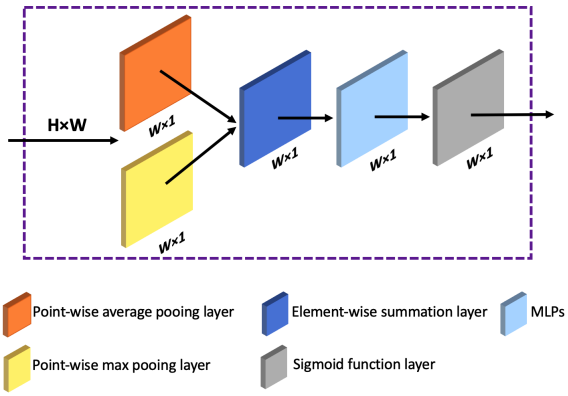

Figure 3.2: The channel-attention block. $H \times W$ represents the input dimensions. Operations are stated inside the box and the feature dimensions are shown after the operation box. Multi-layer perceptron(MLP) has three layers with output dimensions (W, W/16, W).

Due to the occlusion of objects or potential segmentation errors, we might include the pixels that belong to other objects or background. This result could deteriorate the robustness of fusion features. To overcome this problem, our attention module is to refine the fusion features so that it could alleviate the potential problem. Our attention module comprises of two blocks, channel attention block and spatial attention block and this is inspired by CBAM [47]. They are modified to process 1D fusion features used in our network instead of 2D image features originally proposed.

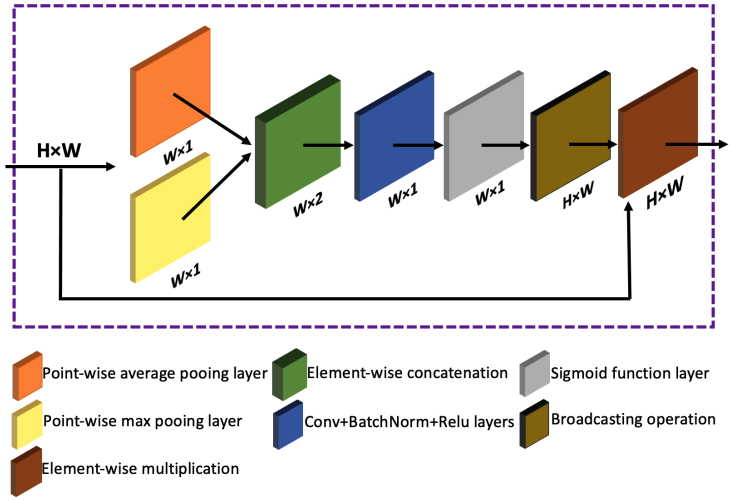

Figure 3.3: The spatial-attention block. The operations inside the box are 1D convolution, batch-normalization, and ReLu function. Broadcasting operation duplicates its input feature map $W \times 1$ for H times to form a feature map with dimension $W \times H$.

Assuming a fusion feature $\mathbf{F}_f$ has a shape $\mathbb{R}^{P \times C}$ where P is the number of pixels, the channel attention block can produce an 1D channel attention vector $\mathbf{M}_c \in \mathbb{R}^{P \times 1}$, the spatial attention block can refine a new spatial attention feature $\mathbf{M}_s \in \mathbb{R}^{P \times C}$. These two blocks are concatenated together as shown in Fig. 3.1. The channel attention block takes the fusion feature as input and generate a channel attention feature $\mathbf{F}^{'}$. These two features are multiplied together and the result is fed to the spatial attention block to generate a spatial attention feature. Again these two features are multiplied together to generate an aggregated feature vector $\mathbf{F}^{''}$. Hence, the overall procedure can be written as follow:

$$\begin{aligned} \mathbf{F}^{'} &= \mathbf{M}_c(\mathbf{F}_f) \bigotimes \mathbf{F}_f \\ \mathbf{F}^{''} &= \mathbf{M}_s(\mathbf{F}^{'}) \bigotimes \mathbf{F}^{'} \end{aligned} \tag{3.1}$$

where $\mathbf{F}^{'}$ is the output from the channel attention block and $\mathbf{F}^{''}$ is the final output that has the same shape with the fusion feature. $\bigotimes$ represents the element-wise multiplication. Broadcast operation to attention map is applied if needed.

Channel Attention Block

The details of the channel attention block are shown in Fig. 3.2. It applies point-wise max-pool and average-pool operations to the fusion feature map respectively, and the resulted descriptors are summed element-wise. A shared multi-layer perceptron network is used to

process the resulted descriptor, which has three neuron layers. To prevent the network's parameters overload, the middle neuron size is set to $W/16$ that is suggested by [47]. The feature map generated by the MLP has dimension $W \times 1$ and is processed by the Sigmoid function to produce the final channel attention map with size $W \times 1$.

Spatial Attention Module

After the first multiplication shown in Fig. 3.1 (to enable the multiplication, broadcasting operation is applied to the channel attention map), the channel attention vector is refined as $\mathbf{F}^{'} \in \mathbb{R}^{H \times W}$. To get the spatial attention feature, max-pool and average-pool operations are applied to generate two features $\mathbf{F}^{'}_{avg}$ and $\mathbf{F}^{'}_{max}$, and both have size $W \times 1$, and they are concatenated. Average-pooling and Max-pooling are commonly used pooling functions. The intuition behind these two pooling functions is that combining the global information captured by average-pool function and the local information captured by max-pool function can have a better performance for our task than using one of them. Here, we use a 1×1 convolutional layer to process the concatenated feature instead of the convolutional layer with kernel size of 7×7, then followed by batch-normalization and ReLu operations. So the spatial attention block is calculated as:

$$\begin{aligned} \mathbf{F}^{'}_{avg} &= AvgPool(\mathbf{F}^{'}) \\ \mathbf{F}^{'}_{max} &= MaxPool(\mathbf{F}^{'}) \\ \mathbf{M}_s(\mathbf{F}^{'}) &= sigmoid(f^{1\times 1}(\mathbf{F}^{'}_{avg}, \mathbf{F}^{'}_{max})) \end{aligned} \tag{3.2}$$

where *sigmoid* denotes the Sigmoid function, which is used to output the normalized feature. Finally the aggregated feature vector $\mathbf{F}^{''}$ is obtained by multiplication, and used to estimate the object pose by the pose predictor.

3.3 Loss Function

In this subsection, we describe the loss function used in our model. we train the loss in a mean square error function as shown below:

$$L_i = \frac{1}{m} \sum_j \left\| (\mathbf{T}x_j - \tilde{\mathbf{T}}x_j) \right\|^2 \tag{3.3}$$

where x_j is the j^{th} point randomly selected from points of object model, and $\mathbf{T}$ is the ground truth transformation and $\tilde{\mathbf{T}}$ is the predicted transformation from j^{th} refined attention features. We also output the confidences of model's predictions, which we would like to utilise to penalise the bad features. So inspired by the DenseFusion [6], we add a regularize term to balance overall prediction. Hence, our final loss function is described as:

$$L = \frac{1}{N}\sum_{i}(L_i C_i - W log(C_i)) \tag{3.4}$$

where N is the total number of sampled refined attention features, C_i is the confidence vector for each sampled refined attention feature and W is the hyperparameter for confidence. During inference, the highest confidence is selected as final output.

3.4 Experiments

In this section, we describe the training details of our network. The network is evaluated on challenging datasets YCB-Video dataset [5] and LindMod dataset [68] . We use a GeForce GTX 1080 Ti graphic card to train our network, which took appx. 300 hours to finish the iterations of 500 epochs on the YCB-Video dataset, and on the LineMod dataset it costs appx. 200 hours to finish 500 epochs. The network is implemented in Pytorch.

3.4.1 Datasets

The LineMod and YCB-Video datasets are two commonly used benchmark datasets. The YCB-Video dataset contains mixed 21 textured and texture-less household objects coming from 92 video sequences. Each frame is annotated with 6D object pose ground-truth. The LineMod dataset has 13 texture-less objects placed on the table in the cluttered background. The datasets were captured by Kinect camera, and each image has its associated depth image and has an object pose annotation. The spilt of training/test sets are unchanged with official datasets.

3.4.2 Training

In the semantic segmentation network for appearance feature learning, *ResNet-18* [28] is used as backbone network, and 4 pyramid levels for pooling are 1×1, 2×2, 3×3, and 6×6. The dimension of geometry feature is set to 1024 and the dimension of appearance feature

is 384, hence, the dimension of fusion feature is 1024 + 384. To predict the pose, we have three independent Multi-layer perceptrons (MLP) applied on the aggregated feature in which each MLP has 5 hidden neuron layers, (1408-640-256-128-4) are the size of hidden layer for the rotation prediction (quaternion), (1408-640-256-128-3) for the translation prediction and (1408-640-256-128-1) for the confidence prediction. In order to prevent over-fitting, we apply data argumentation technique on input RGB patch. For instance, we add some random noises to brightness, contrast, saturation and hue of image of training set. In point cloud, tiny translation error is added. To balance the accuracy and computation, we use Farthest Point Sampling (FPS) algorithm proposed in PointNet to sample 1000 points from the recovered point cloud before feeding it to PointNet. In this way, we can maintain the surface information with limited number of points. The hyperparameter W in equation 4 is chosen as 0.01.

3.4.3 Evaluation Metrics

ADD(S) metrics

To evaluate the network's performance, we use Average Distance of Model Points (ADD) [5] as metric to non-symmetric objects and Average Closest Point Distance (ADD-S) to symmetric objects.

$$ADD = \frac{1}{m} \sum_{x \in M} \left\| (\mathbf{R}x + \mathbf{t}) - (\tilde{\mathbf{R}}x + \tilde{\mathbf{t}}) \right\|_2$$

where $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{t}}$ are the predicted rotation and translation matrices, while $\mathbf{R}$ and t are the ground-truth of matrices. x denotes the points randomly selected from 3D model of object of interest. The prediction of rotation and translation is considered as correct if the score of ADD is lower than a predefined threshold. To evaluate the model's performance on symmetric objects, the ADD-S is used for evaluation. The ADD-S score is calculated as average distance to the closest point.

$$\text{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \left\| (\mathbf{R}x_1 + \mathbf{t}) - (\tilde{\mathbf{R}}x_2 + \tilde{\mathbf{t}}) \right\|_2$$

where x_1 and x_2 are selected from the same 3D model.

We report the area under curve (AUC) for ADD and ADD-S metrics. Also, we set the maximum threshold of both curves to be 0.1m. Beside this, we further test the ADD-S under

threshold 0.01m to illustrate the network's tolerance to small errors. During evaluation, we use ADD metric for non-symmetric objects and ADD-S for symmetric objects.

2D Re-Projection Error

In addition to ADD(S) metrics, we also use the 2D projection metric to quantify the performance of our network. In this way, the object model points are projected to image plane by ground-truth pose and predicted pose. The prediction pose is treated as correct if the average distance of corresponding points is less than 5 pixels. The 2D Re-projection error can be calculated as below:

$$2D\_Reproj = \frac{1}{m} \sum_{x \in M} \left\| \mathbf{K}(\mathbf{R}x + \mathbf{t}) - \mathbf{K}(\tilde{\mathbf{R}}x + \tilde{\mathbf{t}}) \right\|_2$$

where $\mathbf{K}$ is the camera intrinsic matrix.

3.5 Results

3.5.1 YCB-Video Dataset

In this section, we first report the evaluated result of our network on the YCB-Video dataset. We also compare our network with four state-of-the-art pose estimation algorithms (PoseCNN [5], PoseCNN [5] with ICP refinement, PointFusion [37], and DenseFusion [6]). As we can see in Table 3.1, the algorithms are classified into RGB class and RGB-D class. Clearly, the RGB method PoseCNN is lack of accuracy compared with other methods, no matter under which evaluation metrics. We believe this is due to the loss of geometry information. By using the result of PoseCNN as initial estimation, the refinement algorithm ICP can largely improve the performance through optimizing the initial estimation in 3D space. PointFusion [37] and DenseFusion [6] both used RGB image and depth image as their inputs and they can extract appearance and geometrical features for pose estimation. Compared with these two RGB-D methods, our model completely outperform the PointFusion in terms of the performance of individual object or average performance under the ADD-S and ADD(S). Evaluating by ADD-S metric, we lead DenseFusion 0.2% in the performance for all objects, 1.7% under ADD(S). Also, we have more number of highest score objects compared with DenseFusion. It is worth noting that our method has 3 out of 5 best performances on sym-

metry objects (with bold name in Table 3.1). As we know, symmetry objects could cause ambiguity for the feature learning. Hence, we can conclude that our result shows a strong capability of the proposed attention module in learning effective representation from those symmetry objects.

3.5.2 LineMod Dataset

We report the evaluated result of our network on the LineMod dataset. We also compare our network with four state-of-the-art pose estimation algorithms (DenseFusion [6], PoseCNN [5], SSD-6D [34], and PVNet [18]). To achieve a fair comparison, all segmented masks used in these methods are provided by PoseCNN. As we can see in Table 4.1, our method outperforms other methods. Ours refer to the evaluation result using AUC threshold under 0.1m. Our method leads DenseFusion algorithm 3.6% and outperforms PoseCNN nearly 9%. Even we use more strict criteria (ADD-S$<$0.01m), our method achieves an equivalent performance with DenseFusion 94.3% and still outperforms PoseCNN. For the individual object, while DenseFusion has 100% accuracy on *glue*, we achieve the highest prediction on 8 out of 13 objects.

In Table 3.3, the accuracy results by evaluating of 2D projection metric are shown. As DenseFusion does not provide its evaluation result, so we re-trained it to obtain the statistical result shown in Table 3.3. We evaluate the model in three different thresholds(10 pixels, 5 pixels and 2 pixels). Under the condition of 10 pixels, our network has the highest accuracy almost for every objects, except the egg box object, but the gap between them is small(0.1%) and glue object with the same accuracy. In 5 pixel criteria, We see our network has the highest accuracy almost for every objects, except the egg box object, but the gap between them is 0.2%. When the threshold decreases to 2 pixels, both methods' accuracy drop sharply. But our network still has the relative better performance.

We also test our network's performance within small average distance thresholds (ranged 0 - 0.01meter). In this way, we can see how well our model is in the high-precision pose estimation tasks. In Fig. 3.4, we report the accuracy of each object in LineMod with varying threshold. As we can see that until the threshold of 0.006 meters, our network can achieve an accurate prediction ($>$80%). Less than threshold of 0.005, the accuracy curves drop sharply. Note that the object egg box has poor prediction when the threshold is low than 0.07. This

Table 3.1: Quantitative evaluation result on the YCB-Video dataset. Objects with bold indicate symmetry objects. Numbers with bold mean the best performance in comparison.

	RGB		**RGB-D**							
	PoseCNN [5]		**PoseCNN [5]+ICP**		**PointFusion** [37]		**DenseFusion** [6]		**Ours**	
Objects	**ADD-S**	**ADD(S)**	**ADD-S**	**ADD(S)**	**ADD-S**	**ADD(S)**	**ADD-S**	**ADD(S)**	**ADD-S**	**ADD(S)**
002_master_chef_can	84.0	50.9	95.8	69.0	90.9	-	**96.4**	73.1	95.8	71.8
003_cracker_box_vi	76.9	51.7	91.8	80.7	80.5	-	95.5	94.0	**96.4**	95.3
004_sugar_box	84.3	68.6	**98.2**	97.2	90.4	-	97.5	96.5	97.6	97.0
005_tomato_soup_can	80.9	66.0	94.5	81.6	91.9	-	**94.6**	85.4	94.5	85.9
006_mustard_bottle	90.2	79.9	**98.4**	97.0	88.5	-	97.2	94.7	97.9	96.5
007_tuna_fish_can	89.7	70.4	**97.1**	83.1	93.8	-	96.6	81.7	96.8	81.4
008_pudding_box	79.0	62.9	**97.9**	96.6	87.5	-	96.5	93.2	96.4	93.7
009_gelatin_box	87.1	75.2	**98.8**	98.2	95.0	-	98.1	96.6	98.3	97.5
010_potted_meat_can	78.5	59.6	**92.8**	83.8	86.4	-	91.3	83.5	91.7	82.5
011_banana	85.9	72.3	**96.9**	91.6	84.7	-	96.6	83.1	96.4	87.0
019_pitcher_base	76.8	52.5	**97.8**	96.7	85.5	-	97.1	96.8	**97.8**	97.0
021_bleach_cleanser	71.9	50.5	**96.8**	92.3	81.0	-	95.8	90.2	96.0	91.1
024_bowl	69.7	69.7	78.3	78.3	75.7	75.7	88.2	88.2	**88.6**	**88.6**
025_mug	78.0	57.7	95.1	81.4	94.2	-	**97.1**	88.9	96.7	93.0
035_power_drill	72.8	55.1	**98.0**	96.9	71.5	-	96.0	92.8	96.4	95.0
036_wood_block	65.8	65.8	90.5	90.5	68.1	68.1	89.7	89.7	**93.2**	**93.2**
037_scissors	56.2	35.8	92.2	78.4	76.7	-	**95.2**	77.8	90.2	90.2
040_large_marker	71.4	58.0	97.2	85.4	87.9	-	97.5	92.9	**97.6**	92.2
051_large_clamp	49.9	49.9	**75.4**	**75.4**	65.9	65.9	72.9	72.9	73.7	73.7
052_extra_large_clamp	47.0	47.0	65.3	65.3	60.4	60.4	69.8	69.8	**70.8**	**70.8**
061_foam_brick	87.8	87.8	**97.1**	**97.1**	91.8	91.8	92.5	92.5	95.7	95.7
All	75.9	53.7	93.0	79.3	83.9	-	93.1	87.3	**93.3**	**89.0**

situation may be caused by the hard prediction of symmetric object in small tolerance of error of ADD-S metric. In Fig. 3.5, we report the average accuracy of all objects in the LineMod dataset for DenseFusioin and ours. For threshold > 0.03, our curve is in the above of curve of DenseFusion, which means our network has a better performance. Some samples of our estimation results are shown in Fig. 3.6 by projecting their estimated poses back to the image. They provide a clear view on the good quality of our estimation results.

Specifically, we draw the attention maps as shown in Fig. 3.7, where specified region is highlighted as important area for object's pose. The darker the color, the more crucial the area. For instance, in the top row object kettle (object can in Table 4.1 and Table 3.3) is highlighted in its handle area and this region has the highest confidence to object pose. In the second row object driller and fourth row object lamp, their heads are being treated as the parts that have the best estimation for their pose, and we believe this is due to their heads' distinct geometrical information. Furthermore, our model identifies the edge of symmetry object egg box as its focused region which is much reasonable.

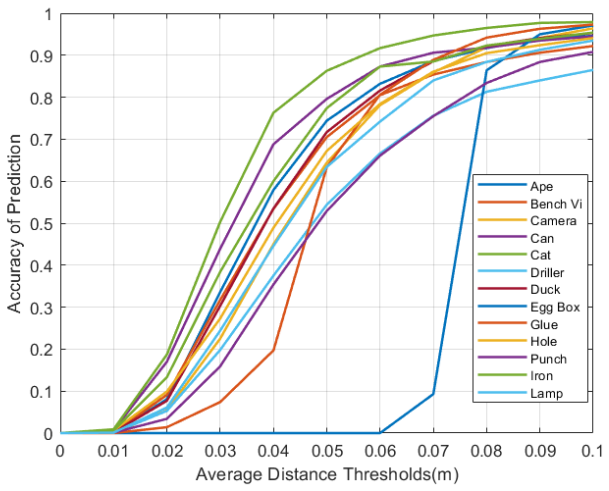

Figure 3.4: Accuracy-Threshold curve for each object in LineMod

3.5.3 Ablation Study

To investigate how our attention modules affect the performance, we test our model with different setups in the LineMod dataset. As shown in Table 3.4, Channel Block and Spatial Block indicate that only corresponding attention block is used in our model, and Channel + Spatial Blocks refers to our complete framework. From the perspective of estimation ac-

Table 3.2: Quantitative evaluation result on the LineMod dataset. Objects with bold indicate symmetry objects. Numbers with bold mean the best performance in comparison.

	RGB-D					
	DenseFusion [6]	**PoseCNN** [5]+**ICP**	**SSD-6D** [34]+**ICP**	**PVNet** [18]	**Ours**	**Ours(ADD(S)< 0.01m)**
ape	92.3	77.0	65	43.6	**94.6**	94.6
bench vi	93.2	97.5	80	**99.9**	96.9	92.2
camera	94.4	93.5	78	86.8	**98.7**	96.4
can	93.1	96.5	86	95.4	**97.3**	94.5
cat	96.5	82.1	70	79.3	**99.3**	97.9
driller	87.0	95.0	73	96.4	**97.1**	86.5
duck	92.3	77.7	66	52.5	**97.0**	95.3
egg box	99.8	97.1	**100**	99.1	99.8	97.1
glue	**100.0**	99.4	**100**	95.6	99.7	97.3
hole punch	92.1	52.8	49	81.9	**97.3**	94.1
iron	97.0	98.3	78	**98.8**	98.6	90.8
lamp	95.3	97.5	73	**99.3**	98.5	95.4
phone	92.8	87.7	79	92.4	**97.7**	93.5
mean	94.3	88.6	79	86.2	**97.9**	94.3

Table 3.3: Quantitative Evaluation result On the LineMod dataset with 2D projection metrics

	ape	bench vi	camera	can	cat	driller	duck	**egg box**	**glue**	hole punch	iron	lamp	phone	mean
DenseFuion(10 pixels)	96.8	92.5	97.7	94.0	97.1	86.5	95.9	**100**	**99.9**	94.9	97.0	95.5	94.6	95.6
Ours(10 pixels)	**98.6**	**95.8**	**98.8**	**97.4**	**99.5**	**95.1**	**98.4**	99.9	**99.9**	**98.2**	**97.8**	**97.5**	**97.6**	**98.1**
DenseFuion(5 pixels)	92.9	85.0	85.0	90.8	95.3	76.2	90.7	**99.7**	99.6	86.9	87.4	91.0	87.6	89.9
Ours(5 pixels)	**94.7**	**90.2**	**94.5**	**94.1**	**98.2**	**85.4**	**94.5**	99.5	**99.9**	**94.0**	**89.8**	**93.3**	**92.2**	**93.9**
DenseFuion(2 pixels)	60.2	48.0	35.3	64.4	66.3	37.0	55.1	67.5	89.1	45.6	**41.0**	56.4	46.6	55.0
Ours(2 pixels)	**63.3**	**51.3**	**49.7**	**67.0**	**77.7**	**37.8**	**61.4**	**68.4**	**91.9**	**56.4**	40.2	**62.2**	**48.4**	**60.0**

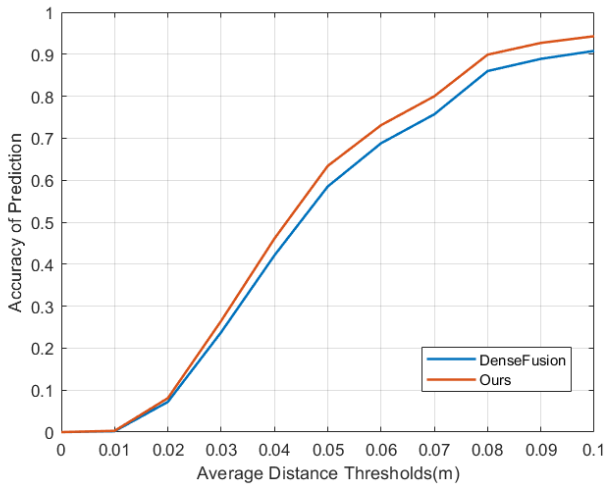

Figure 3.5: Average accuracy by varying average distance thresholds

curacy, the configuration of Channel + Spatial Blocks show the best accuracy in terms of ADD(S) and ADD-S metrics. But it also has the longest inference time (18.2 milliseconds per image) for each input image and the largest number of parameters in memory. The parameters column indicates the parameters for block itself. In the contrary, the System parameters column means the full number of parameters of our model with the existence of corresponding block. As we can see from the table, the Spatial block only has 16 parameters, which is quite tiny compared with the Channel block, and it makes sense that the Spatial block can run with the fastest inference time. Note that the number of parameters in the Spatial block contributes much less to the entire model, because we only have learnable parameters in two convolutional layers as depicted in Fig. 3.3. The Channel block has almost the same system parameters with the Channel + Spatial blocks. In summary, the combination of Channel block and Spatial block do improve the accuracy and they are lightweight compared with the entire model.

We believe that the potential of this attention module could also be used in object pose tracking tasks with a framework of pose refinement that predicts the residual of pose within two consecutive frames. As indicated in Fig. 3.7, our model could focus on some particular regions of the object for pose estimation. This might improve the tracking performance when the occlusion occurs in some scenes.

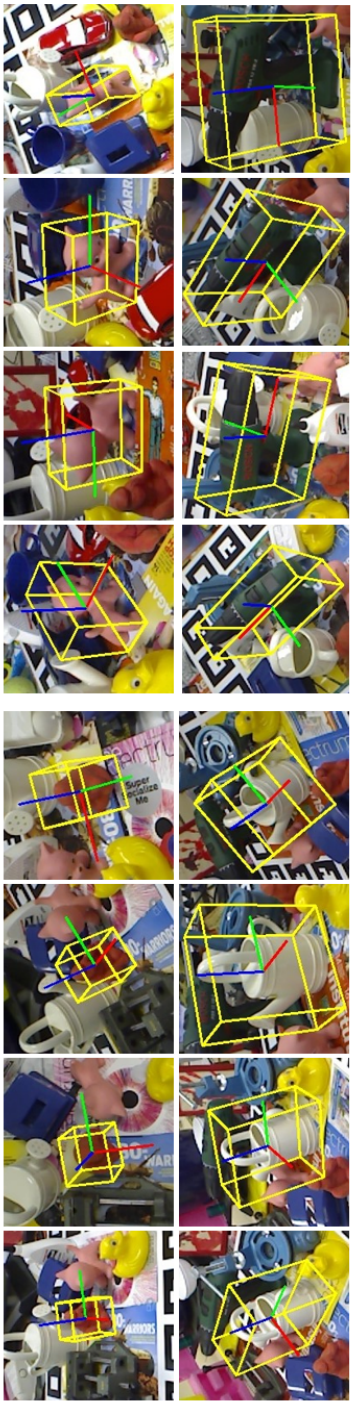

Figure 3.6: Quality results on the LineMod dataset

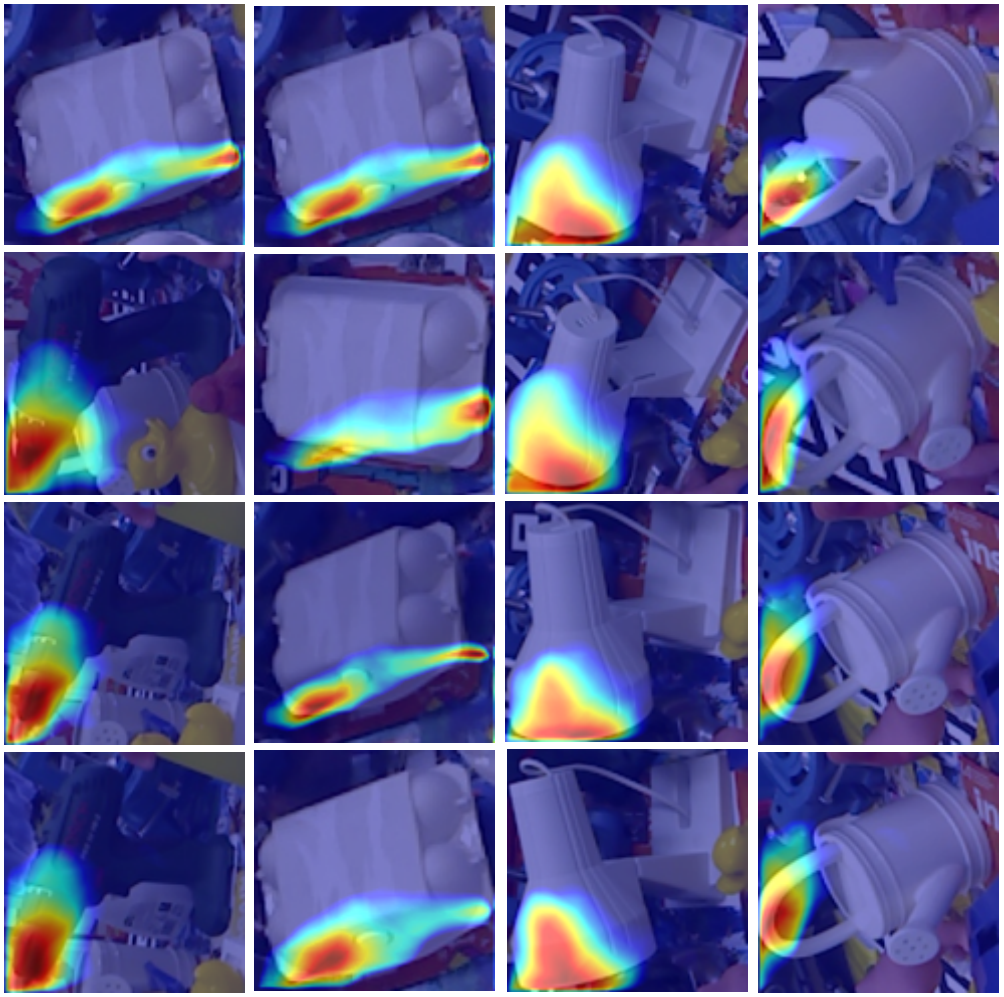

Figure 3.7: Visualization of attention regions

Table 3.4: Ablation study by using either channel block/spatial block or the combination of channel block and spatial block.

	Time(ms/image)	**Parameters**	**System parameters(Millions)**	**ADD(S)**	**ADD-S**
Channel Block	18.1	2184	23.624M	96.5	99.5
Spatial Block	**17.7**	**16**	**23.374M**	97.2	99.6
Channel+Spatial Blocks	18.3	2200	23.624M	**97.9**	**99.7**

3.5.4 Robustness to Occlusion

To explore how robust of our network with occlusion objects, we proposed a *occlusion rate* to reflect how much of an object being occluded. We take p as the total number of pixels of an interested object in ground truth data, and λ as the number of pixels being projected by object model with ground truth pose. Due to self-occlusion of the object, we treat the number of pixels projected in image as $\lambda/2$. So the occlusion rate r can be represented as below:

$$r = 1 - p/(\lambda/2)$$

Therefore, the bigger values of r, the more occlusion of an object. In experiment, we calculated r for each labeled object in the LineMod dataset and averaged them. In Fig. 3.8, we show the performance of our network against different occlusion rates. The blue curve represents our network's estimation accuracy in terms of different occlusion rates. When the r increases, our accuracy remains stable but the curve of DenseFsuion (Orange color) has some fluctuations as r increases.

3.6 Conclusions

In this chapter, we present a network CSA6D that can estimate the 6D object pose from RGB-D image. Both appearance features from RGB image and geometry features from depth image are densely fused together for direct pose regression. Our main innovation includes the use of channel and spatial attention modules to refine the dense fusion feature in order to improve the network performance without adding too much computational burden. Our evaluation results on public datasets show that our network is accurate and robust compared with some existing methods.

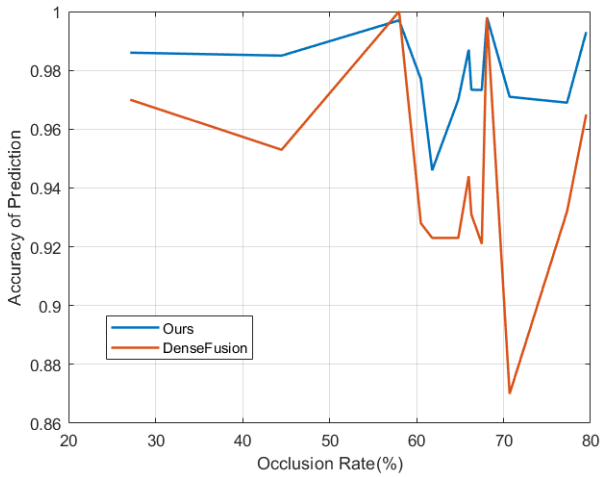

Figure 3.8: Prediction accuracy against object occlusion rate

The attention module is lightweight and efficient, and could be easily inserted into other leaning tasks. We demonstrate that our model can extract features of specific regions for object pose estimation tasks. In our future work, we aim to reduce the computational complexity further in real-time applications. Based on our ablation study of attention blocks, the inference time of our model can be reduced using the spatial attention block only without sacrificing too much accuracy. Also, we believe that making our model (especially the image feature extraction model) lighter could significantly reduce the inference time, which can make it possible to work in real-time applications.

Chapter 4

Siamese Graph-Attention Network for 6D Object Pose Estimation

4.1 Introduction

In the previous chapter, we described an end-to-end learning framework to regress the object 6D pose directly from RGB-D input image. It focused on learning the feature by embedding from different data sources, guided by a distance loss function. Generally speaking, those end-to-end learning methods of object 6D pose estimation have a similar architecture, where a CNN is used to learn the embedding of objects, and the multiple regression layers for regressing the pose of objects. In not an end-to-end learning frameworks, rather multiple regression layers can predict other attributes of objects, for example, 2D keypoints, 3D bounding box and object coordinates, etc. If a network takes RGB-D as input, it normally uses two separate branches of network to process RGB data and depth data, as shown in Chapter 3. To process the depth or depth point cloud data, we tend to extract the sufficient representative geometrical information. Although PointNet exploits the geometry information, local geometrical information is ignored as PointNet only considers individual points. In this chapter, we introduce an edge convolution operation that can leverage point neighbourhood information. We believe that by adding this extra local geometry information the pose features can be extracted more robustly.

Normally, the object pose (Rotation, Translation) is regressed directly by Multi-Layer Perceptron(MLP). Meanwhile, due to the different spaces of rotation and translation, they

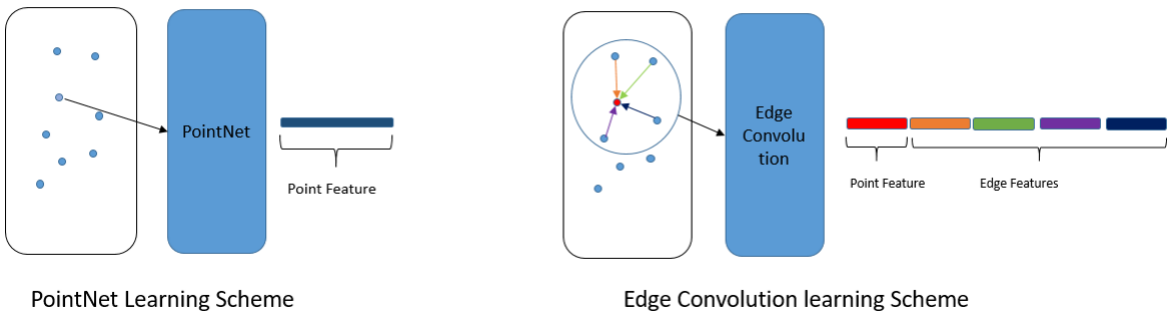

Figure 4.1: Left: PointNet learning, each point is treated as individual. Right: Edge Convolution, each point's neighbours are considered as extra information in learning.

are estimated through different branches. But estimating numerical pose vector from feature space can be problematic, since directly estimating object pose in angle space is not trivial. Therefore, in this chapter, we use a Siamese network to learn discriminated intermediate features from feature space and pose space. An intensive annotation dataset is a key to supervised-learning method for object 6D pose estimation, and the foundation to guarantee the network's performance. However, most of algorithms in object 6D pose estimation, they make use of single input data(RGB, RGB-D, 3D laser scan) to learn the 6D pose with supervised signals. While some methods utilise multiple views, they require a specific designed network architecture to learn the invariant features in supervised or unsupervised ways. For unsupervised ways, the most commonly used learning objectives consider the consistency between the views of two input frames. Therefore, the large discrepancies between the two frames may result in poor network performance. So, in this chapter, we use a Siamese network that shares the same weights, combined with EdgeNet [92], to fully exploit the middle-level features for directly regressing object 6D pose. Our method takes RGB-D data as input, while EdgeNet is used to aggregate the information around each point, and a CNN network to learn the visual features from RGB data. Beside this, we also examine each point independently to determine the feature of each of them. Then, we feed the pixel features, edge features, and point features into an attention network, leveraging the fusion features to achieve the final representation of input data. The final fusion features are used as the input for our pose estimator, which is constructed by MLP layers to regress two vectors ($\mathbf{R}$,$\mathbf{t}$). So, in this way, from the learning process from the layer which records input data to the layer which produces fusion feature, is from image space to feature space, and the learning process

of the pose estimator transforms features from feature space to pose space. Directly using features in feature space to predict the pose in pose space makes the network hard to generalize. So, the accuracy of the prediction is improved through a pose-processing step such as iterative refinement [6]. But in our work, we introduce a Siamese network to make the feature space to closer to the pose space. As illustrated in fig 4.4, we build two identical networks, which share the same weights. These two identical networks input different RGB-D data, in which the objects are presented in different poses. During training, we force the loss function to minimize the error between feature space and pose space. This idea is inspired by the work of [93], which hypothesises that Euclidean distance between samples images from the dataset should remain the same in feature space and pose space. In inference time, we only utilise one network to predict the object pose.

4.2 Related Works

[94] proposed a graph CNN based framework for object 6D pose estimation, in which they simply fuse the node features and appearance features together to regress the object pose directly by MLP layers. This fusion scheme can only leverages the geometrical information efficiently by means of a graph network. [95] proposed a Siamese full flow network to fuse appearance feature and geometric feature, allowing for communication between each encoding layer from image data and point cloud data, and in reverse. FS6D [96] utilises this full flow network as its feature extraction network to learn the representative features from a selection of support images and query images. Then, FS6D uses the linear Transformer to calculate the similarity between support feature and query feature and establish the correspondence. Also, they produce a large shape diversity dataset generated by a physical rendering engine. In a similar fashion to our work, HybridPose [97], adopts two stages to regress the object's pose. In the first stage, it learns the hybrid representation for an object's keypoints, edge vectors and symmetry correspondences. Due to the poor generalization from the hybrid representation to the object pose space by the pose regressor in the second stage, the authors divide the training set into two parts, one for training and another for validation. They introduce critical points into the loss function, to make the pose regression model generalize well into the pose space during testing. [98] uses a Siamese graph neural network to construct the keypoint graph from the combination of synthesis image and real world image.

Due to the lack of labels of keypoints in real world datasets, they assume the geometrical relationship between synthesis and real data remains identical, and train the graph network for the real image by supervising the structure loss.

In our work, we use a Siamese network in a different way, where we still use a graph attention network to learn the robust features from visual data and geometrical data. But, we build up our system under a Siamese architecture, which uses two different images as input. We don't need to design specific layers to learn the invariant pose representations for the objects in input images. We focus on the leaning objective that forces the network to maintain equal distance between their corresponding feature spaces, and the distance between feature space and pose space. In this way, during testing, we improve the network performance in the two stages through our object pose estimation framework, especially as it uses deep regressor to obtain the final object pose.

4.3 Edge Convolution

This subsection, explicitly describes the components of our proposed network, especially the graph CNN we used to process the depth point cloud and how to train our Siamese network. In the below, we called the our graph network as EgdeConv Network. Compared to CSA6D network we proposed in Chapter3, this chapter uses a graph CNN to replace the PointNet in CSA6D net for capturing the local geometrical features and maintaining the permutation invariance. To segment the object of interest and extract the visual features, we still adopt the identical segmentation network architecture as in CSA6D. EdgeConv [92] exploits local geometric structures by constructing a graph (**K-NN graph**) based on each point's K local neighbours.

A point cloud $\mathbf{P}$ can simply be represented as $\mathbf{P} = \{\mathbf{p}_1, ..., \mathbf{p}_n\} \subseteq \mathbb{R}^{F \times n}$, and F initially takes as 3 . We construct a direct graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ by using the K-nearest neighbour graph where vertices $\mathbf{V} = \{1, ..., n\}$ and edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. The edge features are represented as $\mathbf{e}_{ij} = h_\theta(\mathbf{p}_i, \mathbf{p}_j)$ where $h_\theta : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$ is a non-linear function with learnable parameters θ and F' is the output dimensionalities of edge feature. The EdgeConv operator is defined as:

$$\mathbf{p}'_i = \max_{j:(i,j)\in\mathbf{E}} h_\theta(\mathbf{p}_i, \mathbf{p}_j) \tag{4.1}$$

where $\mathbf{p}'_i$ is i-th node output of EdgeConv operator. To explicitly combine the global and

local structures with local neighbourhood information captured by $\mathbf{p}_j - \mathbf{p}_i$, in particular, the operator can be defined as:

$$\mathbf{e}'_{ijm} = ReLU(\Theta_m \cdot (\mathbf{p}_j - \mathbf{p}_i) + \Phi_m \cdot \mathbf{p}_i) \tag{4.2}$$

where Θ_m and Φ_m in equation 4.2 can be implemented as shared MLPs. In the end, the edge feature is selected by applying a channel-wise symmetric aggregation function:

$$\mathbf{x}'_{im} = \underset{j:(i,j)\in\mathbf{E}}{max} \mathbf{p}'_{ijm} \tag{4.3}$$

So, inspired by the EdgeConv operator, the network that extracts the geometrical embedding from depth point cloud is proposed. Figure 4.2 shows of the method through which EdgeConv process point cloud. The input point cloud represented as $N \times 3$ is obtained from the segmentation network. There are four EdgeConv blocks used for learning edge features. The number which follows EdgeConv denotes the output dimensions. The output of each block is aggregated together before being fed into last EdgeConv block. Then, the output of the last EdgeConv block is combined with the features from RGB image, and the features from point cloud, without considering the features of their neighbours. Three multi-layer perceptrons (MLPs) are finally used in the end to predict the rotation $\mathbf{R} \in \mathbb{R}^4$(quaternion), translation $\mathbf{t} \in \mathbb{R}^3$ and confident $C \in \mathbb{R}$ for the prediction.

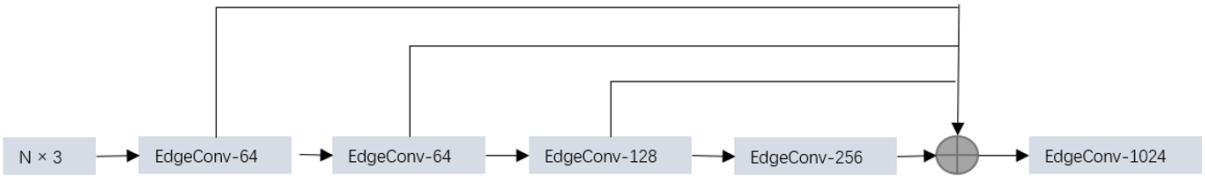

Figure 4.2: EdgeConv Architecture.

4.3.1 EdgeConv Implementation

Figure 4.2 is the EdgeConv architecture for processing point cloud. $N \times 3$ is the size of input point cloud data in which each point contains X, Y, Z coordinates. If EdgeConv IS followed by numbers, it represents the output dimensions of features after processing by EdgeConv operation. Before the last EdgeConv operation, outputs of EdgeConv-64, EdgeConv-64,

EdgeConv-128, EdgeConv-256 are concatenated together. In the end, EdgeConv model will generate the feature with size $N \times 1024$.

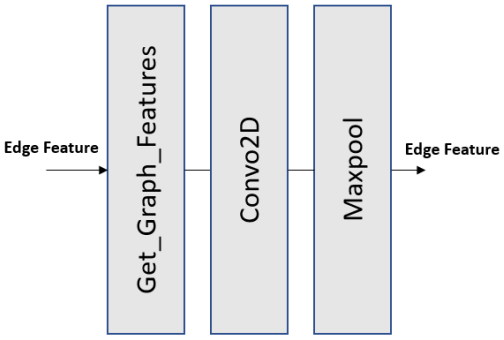

Figure 4.3: Implementation of EdgeConv. It consists of a dynamic updating block for constructing a new graph in each layer, a 2D convolutional operation and a maxpooling operation.

In each EdgeConv step, the point feature will be processed in three steps. Fig 4.3, it shows the implementation details of EdgeConv. First, we will examine the edge features of point's neighbours, and combine their features together. To process the raw input depth point cloud, the coordinates of points are treated as the features of points by $get\_graph\_feature$ operation, in which we recompute the k-nearest neighbours of each point in each layer. Through this updating operation, we can get the different graph from each layer. Unlike the classical graph CNN, where the input graph is fixed after the graph has been constructed. Our experiment shows that this updating process can lead to a better estimation performance compared to a fixed graph. They will then be processed by a 2D convolutional network with kernel size of 1, followed by the maxpooling operation. Through the maxpooling, the local features aggregated by k-nearest neighbours are extracted. When we apply the EdgeConv operation to each point of the input, we can create the edges and obtain the local features for each point, hence, we define the features we obtain as edge features. Combining edge feature and point feature, we can have a more robust representation for the point cloud.

4.4 Model Architecture

Fig 4.4 shows the complete framework for estimating the object 6D pose. Our model uses the same Encoder-Decoder architecture that has been used in CSA6D network for extracting color information. It generates the pixel features with dimension $H \times W \times D$ where H, W and

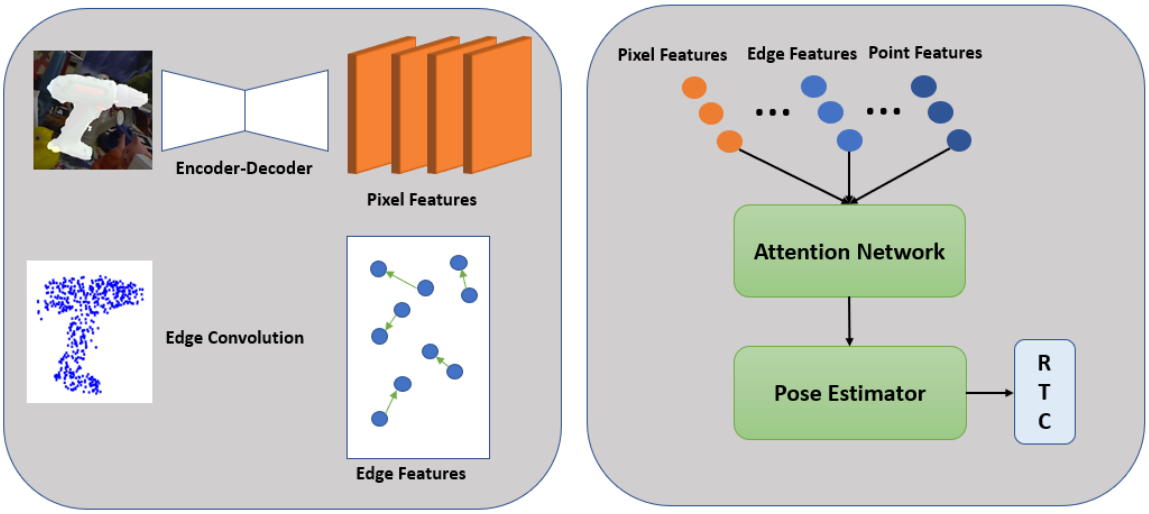

Figure 4.4: Our graph attention network architecture.

D represent the height, width and feature dimensions, respectively. The segmented mask is used to recover the depth point cloud. At the same time, the EdgeCovn network is extracting the edge features and point features simultaneously. To extract point features, the Edge-Conv network treats each point independently,rather than searching the nearest neighbours for each point. Then, we concatenate the pixel features, edge features, and point features together. As in Chapter 3 demostrates, simply stacking the embedding of multi-modalities features results in poor learning performance. So, here, we use a spatial attention to enhance the representation of fused features by weighting the importance of fused features. After applying the spatial attention network, the dimensionality of fused features remain identical. In the end, to regress the pose of object, we construct a pose multiple-layers MLP to regress the rotation vector, translation vector and the confidence of each point.

4.4.1 Features Fusion

In the feature extraction stage, we have extracted the pixel feature and edge feature feature. We also add the point feature to the final aggregated feature, which is obtained by setting $K = 0$ in K-NN algorithm. So, our final feature can be represented as follows: $\mathbf{F}_f = \mathbf{F}_{Pixel} \bigoplus \mathbf{F}_{Point} \bigoplus \mathbf{F}_{Edge}$.

To process the fusion feature, we use the attention module proposed in Chapter 3 to enhance representation of fusion feature. The fusion feature will be processed through the spatial attention module and Channel attention module. Once we get the improved fusion

feature, we use a regressor comprised of 3 branches of fully connected layers to regress the pose vector.

4.4.2 Siamese Network for Pose Regression

As can be seen in most pose regression tasks, especially in the deep learning domain, pose information is learned by using fully connected layers that maps the feature vectors to target vectors, a straightforward but brutal learning approach. In some extreme cases, leaning this mapping function can be challenging. For example, objects with similar appearance can cause ambiguity for the network, as the object with various poses would result in same features. Also, when the object is occluded, the network does not have sufficient features to learn the mapping from feature space to pose space. Even though supervised signal is the most efficient way for the network to achieve its best performance during training, the estimation accuracy will decrease when we apply the unseen data to the network during testing.

This chapter aims to solve this problem by introducing a twin regression network (also known as Siamese network). The twin regression network includes two identical neural networks that share the same parameters, which means we don't need to change the architecture proposed in fig 4.4. The twin regression network tends to learn the interrelationship between the training samples.

In fig 4.5, we show the architecture of twin regression networks that are based on the network proposed in fig 4.4. It comprises of a twin regression network for learning the differences in sample features, and a fully connected layer to estimate pose directly from the fusion feature. We only use the twin regression network in the training stage. First of all, we sample two different RGBD images, in which the objects of interest are presented in different poses. They are then processed by the segmentation network to get the masks of objects. Hence, we can get the region of objects of interest in RGB images and the depth point clouds recovered from depth images. Then we use two separate graph attention networks we proposed in fig 4.4 to obtain the fusion features $f(x_1)$ and $f(x_2)$ for two samples $x_1$and x_2. These two graph attention networks share identical network's parameters. Then two MLP networks take the fusion features to regress the pose p_1 and p_2 for sample x_1 and x_2. As we can see in the fig 4.5, the entire process can be treated as from $x \rightarrow f(x) \rightarrow g(f(x))$.

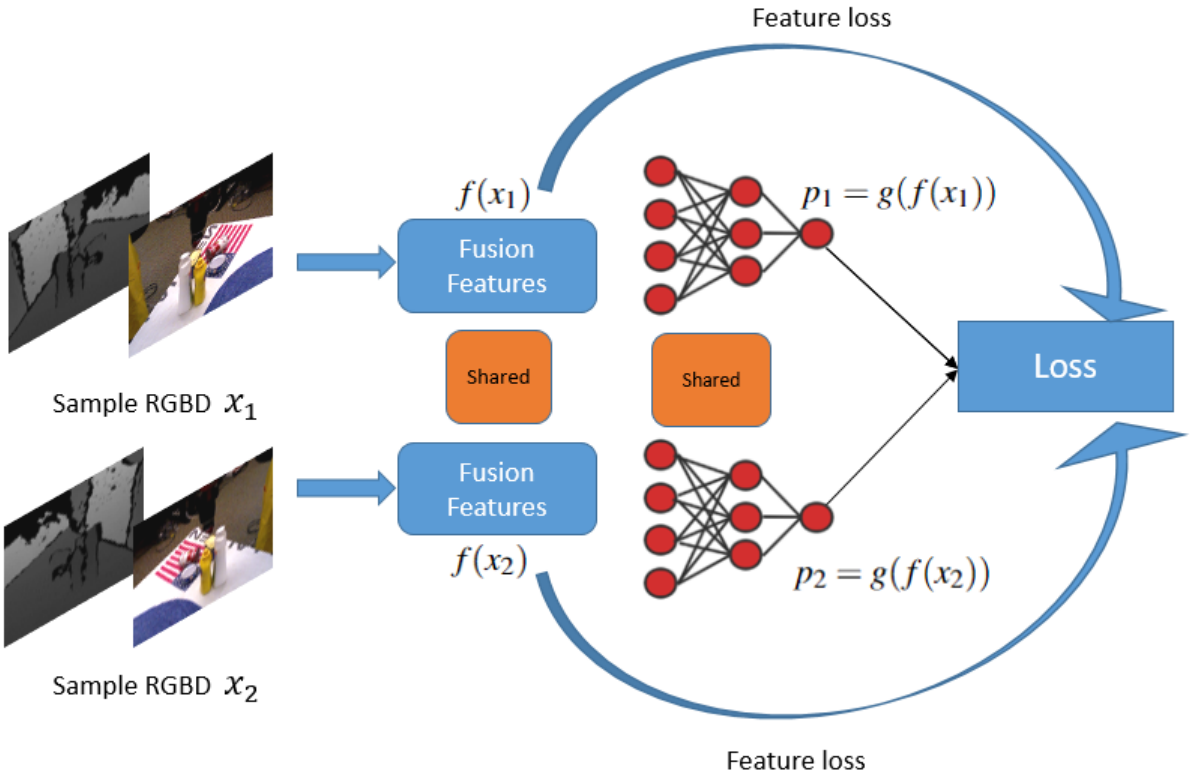

Figure 4.5: Twin regression network architecture.

In the next subsection, we will explain how to introduce them into the loss function to train our network.

4.4.3 Loss Function

To design our loss function, we adopt the hypothesis mentioned in [93], that the Euclidean distance between two samples in feature space should be kept identical in angle space for two same samples during training. Suppose in the training dataset we have training data X_{Train} = (x_1^{train}, x_2^{train},...,x_n^{train}), with their labels Y_{Train} =(y_1^{train}, y_2^{train},...,y_n^{train}), in which X^{train} consists of RGB image and depth image, and Y^{train} has the ground-truth data (Rotation and translation) for the objects in the X^{train}.

To train the twin regression network, a pair of data (x_1^{train}, x_2^{train}) is sampled from the dataset. In equation below, we use x_1, x_2 without superscripts to represent the sampled data. The twin network training process can be benefit from the objects with various pose. To collect the samples from dataset, we set the step as 10.

Our loss function is comprised of two parts, L_{pose} and L_f. We first define L_p as the loss of pose predictions of the twin network.

$$L_p = \frac{1}{m}\sum_{j=1}^{k}\sum_{i=1}^{m}\left\|(\mathbf{T}_j x_i - \tilde{\mathbf{T}}_j x_i)\right\|^2 = \frac{1}{m}\sum_{j=1}^{k}\sum_{i=1}^{m}\left\|y_j x_i - g(f(x_j))x_i)\right\|^2 \tag{4.4}$$

where $\mathbf{T}_j$ and y_j are the ground-truth pose, $\tilde{\mathbf{T}}$ and $g(f(x_j))$ are the predicted poses from last MLP layer that utilises the attention features. m is the number of points sampled from the object model. k is the amount of sample data we collected from the dataset. Essentially, equation 4.4, uses the same loss function as equation 3.3 from chapter 3. The object's model is transformed by the ground-truth pose and estimated pose, and their average distance is calculated. The difference here is we have more than one form of input data. We also predict the confidence score C_i of each attention feature for balancing the entire prediction. Therefore, the final loss of pose regression L_{pose} is defined as:

$$Lpose = \frac{1}{n}\sum_{i}^{n}(L_p C_i - Wlog(C_i)) \tag{4.5}$$

where n is the number of attention features and W is the hyper-parameter to balance left term and right term in equation 4.5. In the stage of testing, the predicted pose with highest confidence will be treated as the final output of the network.

For L_f loss, we force the network to maintain the same distance between its feature space and pose space. In other case, as shown in fig 4.5, the distance between attention features $f(x_1)$ and $f(x_2)$, and the distance between labels of pose y_1 and y_2 are used as the learning objective in L_f. Here, we use $L2$ distance to calculate their difference. In order to make the training process consistent, we normalize the output of last layer of attention network. Also, the output from pose regression layers is normalized. If quaternion is estimated by the pose regression layers, it already has unit norm. Finally, we can define L_f in the format as:

$$L_f = \frac{1}{k}\sum_{i}^{k}\left\|\underbrace{\left\|f(x_{i,1}) - f(x_{i,2}))\right\|^2}_{\text{feature differences}} - \underbrace{\left\|y_{i,1} - y_{i,2}\right\|^2}_{\text{pose differences}}\right\|^2 \tag{4.6}$$

where k is the number of samples. $f(x_{i,1})$ and $f(x_{i,2})$ are the normalised attention features. $y_{i,1}$ and $y_{i,2}$ are the corresponding labels for samples. In equation 4.6, the training loss consists of loss in feature space and loss in pose space. Here, we explain the case when symmetrical objects are encountered during training. As texts explaining in equation 4.6,

the term $\left\|y_{i,1} - y_{i,2}\right\|^2$ measures the differences between the pose of sampled objects. For the large different pose, it is likely that network will generate identical features, which are reflected as the result of $\left\|f(x_{i,1}) - f(x_{i,2}))\right\|^2$ to close in zero. To alleviate this ambiguity, the loss L_f will enforce function f to learn different representations for objects $x_{i,1}$ and $x_{i,2}$ in order to minimize itself. When the sampled objects have similar pose, the loss L_p can enforce the network to predict estimated $\tilde{\mathbf{T}}_j$ to close in its ground-truth pose $\mathbf{T}_j$. In the end, to balance this two situations, we combine L_{pose} and L_f to define our final loss function:

$$L = \alpha L_{pose} + (1 - \alpha) L_f \tag{4.7}$$

where α is the hyper-parameter used to determine the importance of the terms in equation 4.7.

4.4.4 Training Details

This subsection explains the details of constructing our network and how to train the network. As in the Chapter 3, a pre-trained Mask-RCNN [26] is used to segment the object of interest. PsPNet [91] is adopted to process the segmented images, and to extract the visual features. We use camera intrinsic to recover the depth point cloud from the mask of object and its corresponding depth image, and we sample 500 points from the depth point cloud, forwarding to EdgeNet to extract edge features and point features, where we set $K = 8$ to build the $K - NN$ graph. In order to obtain the features of individual point itself, it is not necessary to search the neighbours of each point. In the fig 4.6, we show the size of each feature from different branches. Note that when we build our twin regression network, we just duplicate the network as shown in fig 4.6. The input RGB-D data we take, for example from LineMod data, is the size of $480 \times 640 \times 4$ including 3 channels RGB image and 1 channel depth image. For simplicity, the segmentation network doesn't show in fig 4.6. Suppose we already have the mask of object of interest, we can plot a tight bounding box around the object with the size $H \times W \times 3$. Furthermore, we can recover the depth point cloud from its corresponding mask regions, where we sample 500 points, expressed as $[500, 3]$. In fig 4.6, the *pixel features* denote the feature obtained from PsPNet, which are represented by yellow block. The *pixel features* will be encoded with the dimensionalities $32 \rightarrow 64 \rightarrow 128$ without changing the height and width of the cropped image. The processing of *edge fea-*

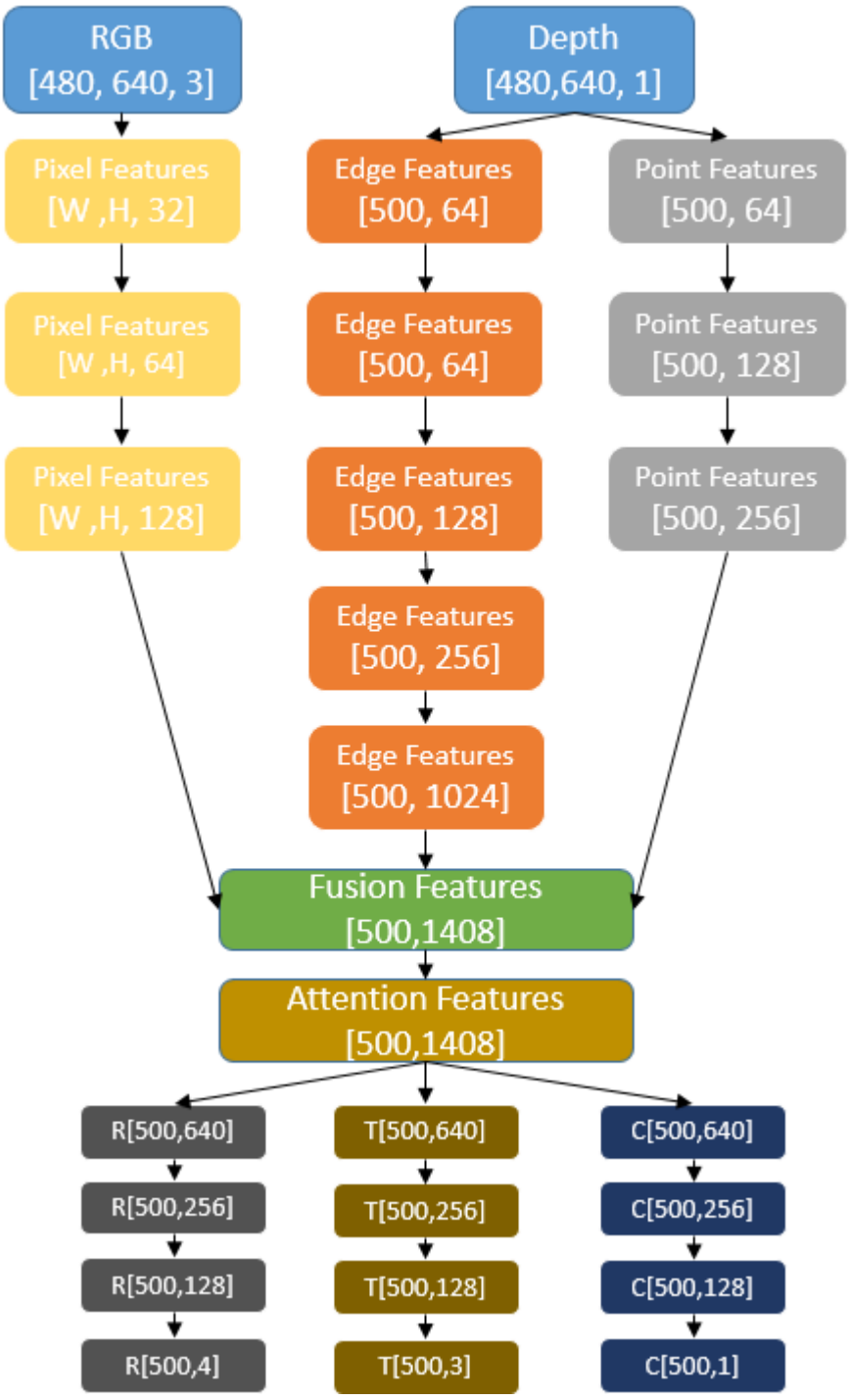

Figure 4.6: The flow of feature dimensions of our proposed graph attention network.

tures is shown in the orange color block. This block takes the recovered point cloud as input and constructs a KNN graph for each orange block. The *edge features* are calculated by the EdgeNet which is illustrated in fig 4.3. In the end, we get the *edge features* with the size of $[500, 1024]$. The grey blocks indicate the processing of *point features*, and it has the final output as $[500, 256]$. To fuse the different sources of data, we stack the *pixel features*, *edge features* and *point features* together and can obtain the fusion features with the size of $[500, 1408]$. Then, the attention network refines the *fusion features* to learn the most representative features for further object 6-dof pose regression. The implementation details of the attention module can be found in subsection 3.1.2 of Chapter 3. The last part, the 3 branches MLP with 4 layers, are used to regress rotation vectors(with quaternion), translation vectors, and confidence vectors.

During training, we duplicate our graph attention network, to make it taking 2 pairs of RGB-D input data simultaneously. We set the hyper-parameter α in equation 4.7, as 0.5 in training. Our network is implemented using PyTorch package and is trained on the machine with NVIDIA GeForce GTX 1081 Ti and Intel® Core™ i9-7900X CPU @ 3.30GHz × 20 CPU. The network is optimized by Adam algorithm [99] and is trained by the learning rate $1e-4$.

4.5 Experiment Results

In this section, we demonstrate the effect of the experimental results on the LineMod dataset. We show the quantitative results based on ADD-S metric and pose root mean square error(RMSE) for rotation and translation. We use these results to prove that our network can achieve the best performance compared to networks which adopt a similar training scheme to ours. We also demonstrate that the performance of the the proposed graph attention network can be improved by training as twin regression network, especially for symmetry objects.

Evaluation Metrics. To evaluate the performance of the network, we use ADD(Eq. 2.24) and$ADD-S$(Eq.2.25) which are commonly used metric for object 6D pose estimation. Also, the pose root mean square errors($RMSE$) are reported, where the average errors for Euler angles ($RMSE_R$) and translation vector ($RMSE_t$) are calculated. In table 4.2, we use R_{err} and t_{err} in short.

In table 4.1, we show the effectiveness of performance of our proposed network on

Table 4.1: Quantitative evaluation result on the LineMod dataset for Graph Attention Network with Twin regression network. Objects with bold indicate symmetry objects. Numbers with bold mean the best performance in comparison.

	RGB-D					
	LineMod [32]	LCHF [100]	Balntas et al [101]	Balntas et al * [101]	**Ours ADD-S**	**Ours ADD**
ape	32.01	35.51	53.26	51.61	63.5	62.5
bench vi	28.64	46.92	52.89	51.70	75.3	76.5
camera	-	-	59.73	57.21	69.8	72.5
can	24.45	35.25	51.59	49.84	81.0	82.0
cat	33.74	35.51	57.46	56.66	82.7	81.9
driller	21.22	43.38	46.71	45.16	59.7	60.0
duck	30.16	37.50	51.58	50.51	62.3	66.4
egg box	-	-	59.29	58.20	99.9	76.1
glue	-	-	55.51	53.14	99.4	73.2
hole punch	32.01	43.33	55.84	57.51	66.4	68.2
iron	26.82	38.80	60.31	58.32	81.9	82.4
lamp	22.81	36.59	57.91	57.76	88.9	90.0
phone	23.25	32.01	61.35	58.66	76.4	77.8
mean	27.51	38.48	55.64	54.32	**77.5**	**74.5**

LineMod dataset, compared to the methods that focus on learning the RGB-D descriptor for object 6D pose estimation. LineMod [32] samples the object from different viewpoints, and generates features from color gradient, with surface normal as the template. The object to be estimated is matched with the templates to get its pose. LCHF [100] combines the LineMod and Hough Forests to do the pose estimation and alleviates the degraded network performance in the existing occlusion and background clutter. Balntas's work [101] proposes a deep RGB-D descriptor leaning framework, which can realise the object recognition and object pose estimation tasks. They implement a similar scheme to our proposed model, but they consider the negative samples in order to distinguish the different objects from the dataset. As we built a segmentation network in our pipeline, we do not necessarily consider these negative samples. The records of performance are obtained from the published work of [101]. The star in Balntas's method indicates the performance of using direct regression loss, otherwise the only differences considerred would be feature space and pose space.

In table 4.1, the first four columns measure the pose estimation accuracy of the methods by the percentage which correctly matches their closest template. The results demonstrate that the performance of our network exceeds others. We achieved 77.5% average prediction accuracy on ADD-S metric, and 74.5% average prediction accuracy on ADD metric. We believe this is due to the well-designed network's architecture and learning metric.

In table 4.2, we evaluate the performances of our network by training with different loss

	ape		benchvise		cam		can		cat		driller		duck		**eggbox**		**glue**		holepunch.		iron		lamp		phone		average	
	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}	R_{err}	t_{err}
L_{pose}	16.40	0.885	12.46	0.85	14.12	0.89	13.42	0.86	14.79	0.88	17.15	0.85	15.44	0.88	94.14	0.89	89.22	0.87	15.45	0.90	13.57	0.89	12.49	0.86	12.97	0.87	26.50	0.87
$L_{pose}+L_f$	17.66	0.88	16.57	0.85	16.53	0.89	17.80	0.86	15.39	0.88	30.36	0.85	15.85	0.88	19.12	0.89	24.63	0.87	23.16	0.90	16.86	0.89	13.28	0.86	17.69	0.87	18.83	0.88

Table 4.2: Ablation study that compares the performance of combination of L_{pose} and $L_{pose}+L_f$, evaluated on the LindMod dataset by calculating rotation and translation errors, R_{err} and t_{err}.

functions. Here, we test two varying loss functions, L_{pose} and $L_{pose}+L_f$, respectively. When training the network solely with L_{pose}, the network just learns the features, guided by the difference between the predicted pose and ground truth pose, as illustrated in equation 4.4. However, when combined with $L_{pose}+L_f$, the relationship between the features and pose are considered, which can help the network to learn more representative and robust feature descriptors from RGB-D data. As we can see in table 4.2, two kinds of metrics are reported, rotation error R_{err} and translation error t_{err}, respectively. The network only achieves an average 26.50 degrees of R_{err} when $L_{pose}isused$, and an average 0.87m of t_{err}. In terms of estimating the rotation and translation, the network makes a more precise estimate of the translation vector, which can achieve 0.87m of t_{err} on average when training with L_{pose} loss function. We note that the accuracy of the estimation of the translation vector are almost identical within two different loss functions. For the sake of simplicity, we only report two digits after the decimal. For the rotation estimation, we can see that in the training with $L_{pose}+L_f$, the rotation error has been decreased from 26.50 degrees to 18.83 degrees. In particular, the network makes extensive errors when estimating the symmetry of the objects **eggbox** and **glue**, when learned by $Lpose$. Guided by our proposed loss function, we can see that the R_{err} error has been reduced from 94.14 degrees to 19.12 degrees for **eggbox** object, from 89.22 degrees to 24.63 degrees for **glue** object.

4.6 Conclusion

In this chapter, we use the graph convolutional network to improve the geometrical embedding representation from RGB-D data in the object 6D pose estimation task. It takes the neighbouring information of each individual point into account by constructing a K nearest the neighbouring graphs, followed by an edge convolutional network to dynamically update the geometrical features. To learn a robust fused feature from different data format, we utilise the attention scheme to select the representative fusion features. Finally, the pose is regressed by the MLP layers. In order to alleviate the ambiguity pose estimate caused by the symmetry

object, we construct a Siamese network, which enforces the feature learning guided by the pose differences. The quantitative results evaluated on LineMod dataset demonstrate that the precision of rotation estimation can be improved through training with our proposed loss function.

Chapter 5

6D Object Pose Tracking with Optical Flow Network for Robotic Manipulation

In this chapter, we propose a novel 6D object pose tracking framework for robotic manipulation tasks. The framework takes a RGB-D video stream as input and outputs a 6D pose estimate corresponding to each frame for the interested object to be tracked. The novelty lies in the pose change estimation where we leverage a segmentation network and an optical flow network to estimate the pose change between previous and current frames. The final 6D pose estimate is the multiplication of the 6D pose matrix in previous frame and the pose change. Unlike most tracking networks, our pose tracking mdeol does not require any object 3D model as auxiliary input. We take two consecutive frames as the input and estimate their optical flow map by using a pre-trained optical flow network. Our framework is a keypoint based estimation method. The estimated flow map can extract the temporal motion information that can be used to generate keypoint candidates. Then an iterative keypoint refinement scheme is used to validate the selected keypoints. Our experimental results show that our framework can outperform some existing works or achieve comparable results in three selected datasets.

5.1 Introduction

Object 6D pose tracking, which is the process that continuously predicts the motion of object(normally rotation and translation vector, velocity and the sizes of object included in some of the cases) in the time sequences. It has been considerably used in the computer vi-

sion and robotics communities, where the tracked objectives can be human beings, animals, objects, and various signals. For instance, in the scenario of health care for older people, understanding and tracking their movement is the key of preventing them from the harm of falling and some other emergencies. Yan et al [102] proposed a graph neural network based tracker to track the movement of body skeletons, by extracting the joints' features from spatial and temporal dimensions. In intellectual agriculture, where the drone can be used for spraying pesticides in order to avoid the human involving from the toxic substances. Therefore, a precise tracking of the trajectories of the drone is the important step to accomplish this gold [103] [104]. In the emerging domain of AR/VR, the high accuracy of pose estimation to objects and players are crucial for the good experience of those equipments. To render a realistic environment, a robust and precise pose estimate or tracking framework is essential for the reconstruction of such environments [105] [106].

Object 6D pose estimation from a single frame is the process where the estimated pose is produced only by current frame while object 6D pose tracking usually considers spatial and temporal information from consecutive frames to predict or update the pose estimation. In recent years, we have seen many efforts to address various 6D pose tracking problems, but some scenes still pose significant challenges, such as clutter environment, occlusion, light condition variations, etc. Deep learning based methods have been explored and have shown the superiority for dealing with the challenging scenes. Learning based methods use large amount of labelled data to train the networks in order to optimize the network models that can regress the numerical 6D pose. Learning to select the keypoints of interested object is one of the popular learning-based methods, in which the selected keypoints are learnt from input frame and then the pose information is extracted from them.

In this chapter, we proposed a novel tracking framework that leverages a segmentation network and an optical flow network to achieve the object 6D pose tracking. Unlike the most tracking networks, our framework does not require any object 3D model as auxiliary input. We take two consecutive RGB-D frames as input and estimate their optical flow map by using a pre-trained optical flow network. The flow map records the motion information of interested object that can be used to generate the keypoints. The optical flow estimation is robust to the noise in object appearance or some occlusions. We calculate two optical flow maps: forward flow and backward flow between previous and current frames. Then

the consistency between these two flow maps are used to remove the outliers. We use the segmentation network to generate an object mask. From the consistency map and object mask, the keypoints are selected. Furthermore, we propose an iterative keypoint refinement scheme to validate the selected keypoints.

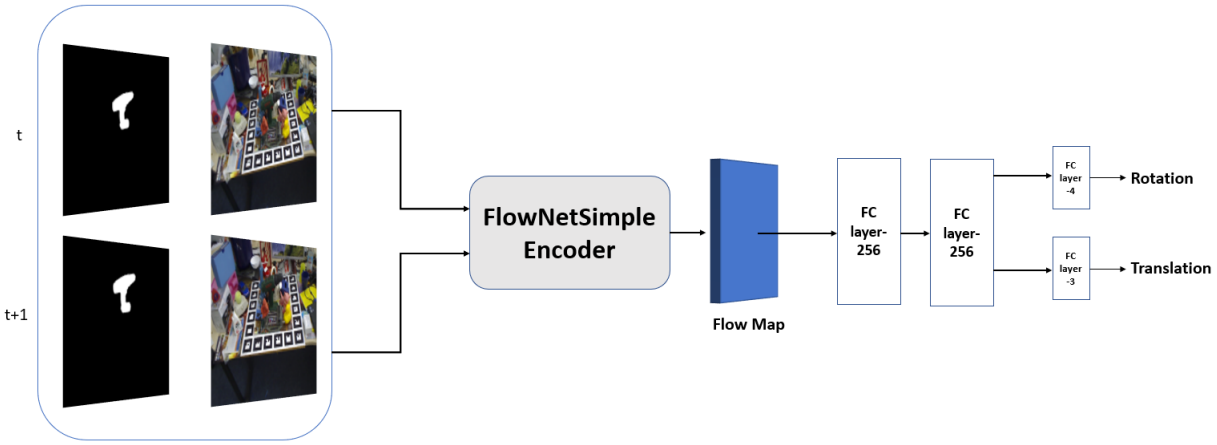

Figure 5.1: The tracking model that only utilises flow features to predict object 6D pose.

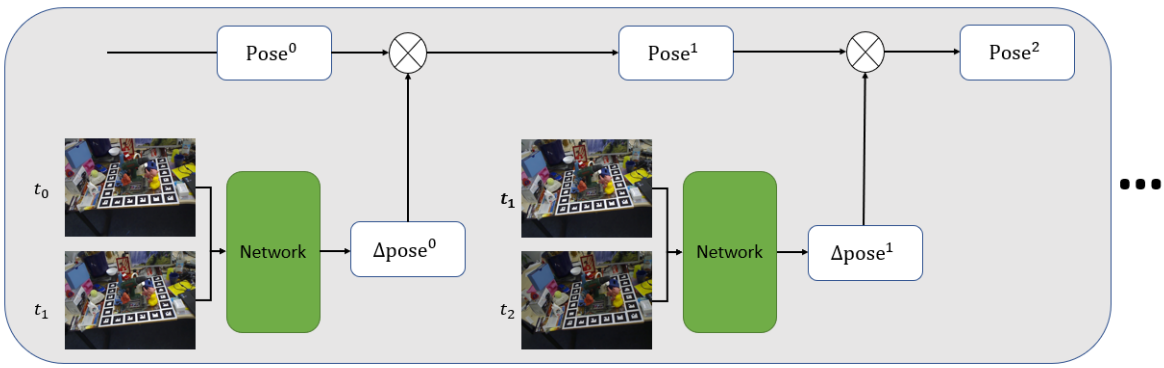

Figure 5.2: A tracking pipeline. Given the initial object pose $Pose^0$ the network was used to predict the pose change $\Delta Pose$

5.2 Relative Works

In this section, we review the works of object 6D pose tracking that are based on learning approaches, especially we focus on the reviewing of keypoint-based, category-level methods. Our method is belong to keypoint-based tracking, but it can also be used in the category-level objects tracking without learning the specific features to distinguish various kinds of object.

5.2.1 Keypoint-based Object Pose Tracking

Learning-based keypoint tracking replies on unsupervised or semi-supervised objectives to find keypoints that can establish a geometrical correspondence for the previous view and current view. So, the pose is not being learnt in an end to end manner. In the work of [107], Wang et al proposed a Anchor-based keypoints scheme, which can be used for tracking the category-level objects. They designed a 3D cuboid around the depth point cloud that recovers from the segmented mask, and the distance-weighted values from point cloud to the anchors on the 3D cuboid are treated as the features for each anchor. Then, a self-supervised network is used to estimate ordered keypoints for predicting the pose transformation. Wen et al [61] introduced a tracker that leverages a network to produce category-level semantic keypoints. An initial relative transformation, then can be obtained from the keypoints registration. In the backend of their framework, a pose graph [108] is utilised for refining the initial pose. The work [109], is the combination of keypoint estimation and undertainty estimation for a known category object, where the estimated 2D projected keypoints derive from the object 3D bounding box and Bayesian filter is used for filtering out label and sensor noises. The aforementioned methods are affected by the quality of visual feature of input data. For tracking a texture-less or non-texture objects, it's hard to learn the reliable presentation for those objects. Our method can overcome these limitations well.

5.2.2 End-to-End Learning Object Pose Tracking

Differing from the above methods, this end to end tracking directly regresses pose from the input data. In the work of Wen et al [62], they utilised the combination of real data and synthetic data to train encoder-decoder-like network to regress object pose via Lie Algebra representation. Li [110] proposed an iterative matching scheme which also be applying in pose tracking. They use an ligh-weight optical flow network as a backbone network to learning deep features, which then be passed to two separated fully connected layers to predict rotation and translation, respectively. Unlike the way of using optical flow network in our study, [110] needs a Zoom-In pre-process step to enlarge the details of real image and rendered image for the purpose of keeping enough visual details, so that optical network can learn the robust feature representation. The work [111], proposed by Piga et al, is the method performing object pose tracking through UKF [112] from object mask and ob-

ject CAD model. They use recovered point cloud as the measurement for the measurement model of UKF to update the posterior distribution of object pose. Deng et al [60] utilised Rao-blackwellized particle filtering to track the object's pose distribution, where the estimation of rotation distribution is conditioned on the translation estimation. The particles is firstly propagated to predict likely RoI(Region of Interest) patches, followed by a pre-trained autoencoder to update the possibility of rotation's posterior distribution. The accuracy of this method is inherently limited by the sampling space of particle filters as it rotation space is discretized into a fixed distribution. While the particle distribution can be precisely, the burden of computation increases.

5.2.3 3D Dense Scene Flow

In this section, we summarize some related experiments, that inspire us to propose the final framework. These experiments is designed around utilising an optical flow network to perform object 6D tracking from RGB or RGB-D input.

Inspired by the work of [110], we use the fully connected layers to regress pose change directly from the features that learned from an optical flow network, and this pose change will be used for the next frame for tracking. This process is shown in the Fig. 5.1. The observed RGB images and corresponding masks for interest objects at timestamp t and t+1 are feed to the model. FlowNetSimple will generate an optical flow map ($8 \times 10 \times 1024$) that encoded the relative motion information. After optical flow estimation, we use fully connected layers to regress pose, where translation and rotation are regressed by two separate FC layers.

In Fig. 5.2, we show our tracking pipeline combined with the network of fig 5.1. The initial pose value $Pose^0$ is given for observed image at t^0. The network tasks images at t_0 and t_1 to estimate a relative pose change for t_1. So we can express the pose of considered object at t_1 as $Pose^1 = Pose^0 \cdot \Delta Pose^1$. After K time steps, the pose therefore can be express as:

$$Pose^k = Pose^0 \cdot \Delta Pose^1 \cdot \Delta Pose^2 \cdots \Delta Pose^k \tag{5.1}$$

But this kind of networks suffer the challenging to predict the precise pose parameters. Directly learning $\Delta pose$ from image space to pose space can be tough. It can be shown that such network,like **steerable CNN** [113] [114] with properties of equivariance and equivarient will have a better generalisation in this directly pose estimation task. However, this

network can not guarantee the real-time performance.

As our gold is to predict the state of object in 3D space, therefor we conducted the experiments that estimate the 3D flow(scene flow) from depth point clod. Scene flow can provide the understating of dynamic environment, like moving direction of the point in previous frame to next frame. It's 2D projection in image frame is the optical flow. In this study, we mainly used the deep-learning based methods to obtain the scene flow. FlowNet [53] and FlowNet2 [54] are two CNN networks to do that, showing the high accuracy and efficiency. However, those two networks mentioned above can not deal with the irregular data, for example 3D point cloud. Hence we utilised the FlowNet3D [115] which aims for the 3D scene learning.

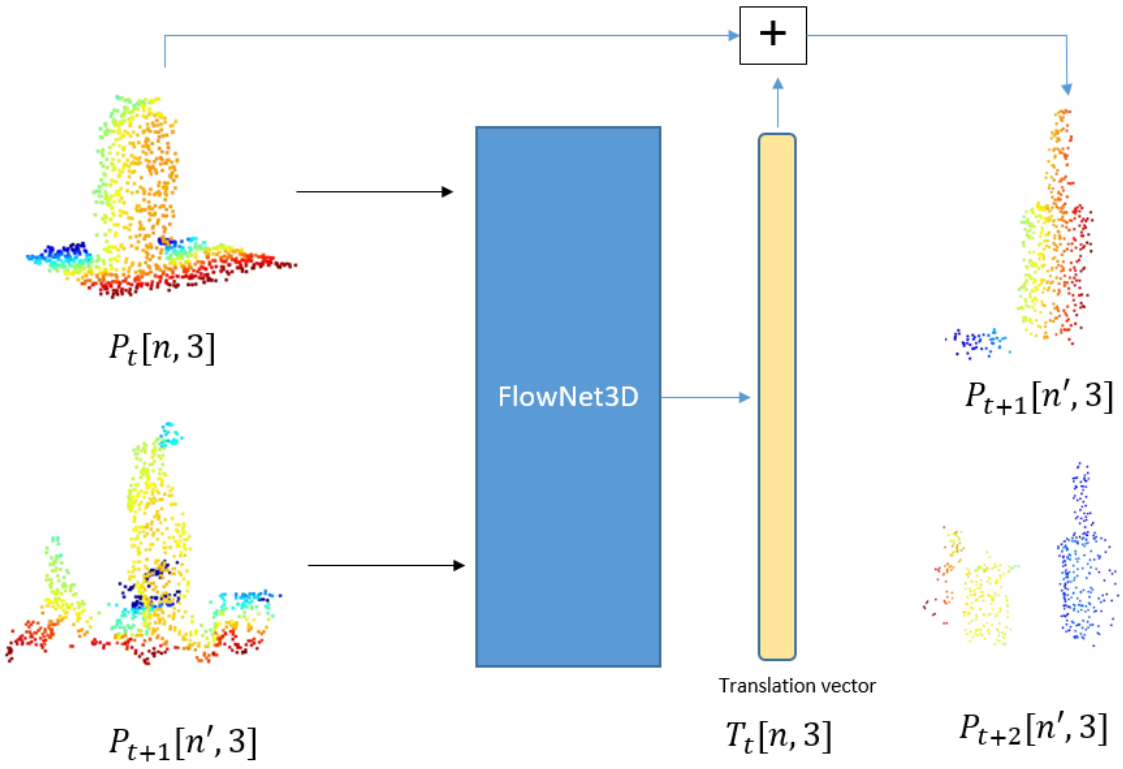

Figure 5.3: 3D scene flow prediction using FlowNet3D

Fig 5.3 shows the tracking architecture based on FlowNet3D. $\mathbf{P}_t$ and $\mathbf{P}_{t+1}$ are two consecutive point cloud sets. The FlowNet3D is going to predict the translation vector with respect of $\mathbf{P}_t$. Hence, the relative transformation can be estimated by optimising a least square problem.

$$arg\min_{\mathbf{T}}||\mathbf{P_t}\mathbf{T}-\mathbf{P}_{t+1}||^2 \tag{5.2}$$

Where **T** is the relative transformation, which includes rotation and translation. To train this network, we simply construct a distance loss function that try to minimize the average distance between two point clouds.

$$L_{dis} = \frac{1}{n}||\Sigma^n(\mathbf{P}_t^n\mathbf{T}^n) - \Sigma(\mathbf{P}_{t+1}^n)||^2 \tag{5.3}$$

Where the n is the number of the points in point cloud.

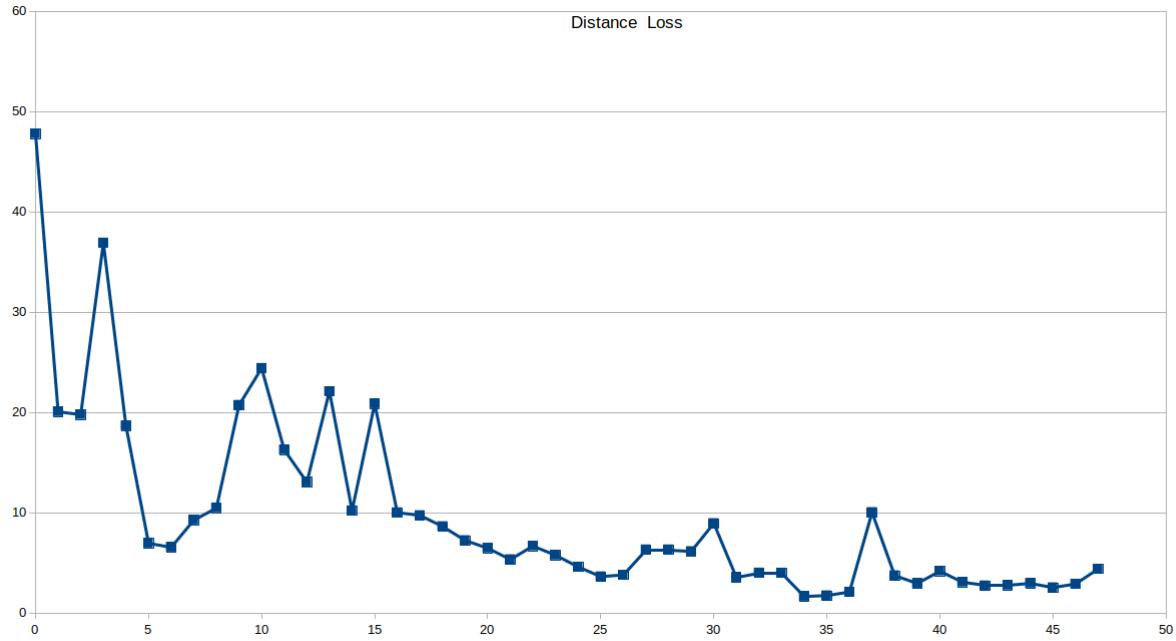

Figure 5.4: Training loss of L_{dis} .

In Fig 5.4, we plotted the loss of L_{dis} function in the first around 50 epochs. We can see that the loss are deceasing gradually after 15 epochs. It turns out that loss function starts convergence. However, the we can not reach the minimal of the loss function as function was stuck in the sub-minimal region. Actually, learning such distance loss function can be real challenging. First of all, the network must learn the correspondences from two two point cloud. Unfortunately, the point cloud was recovered from the mask of interest object and its depth map where both have nosily data due to imprecise segmentation and sensor noisy.

Based on the training analysis of L_{dis} function, we imposed more constraints on the network. We constructed a loss function that consider the distance and pose prediction su-

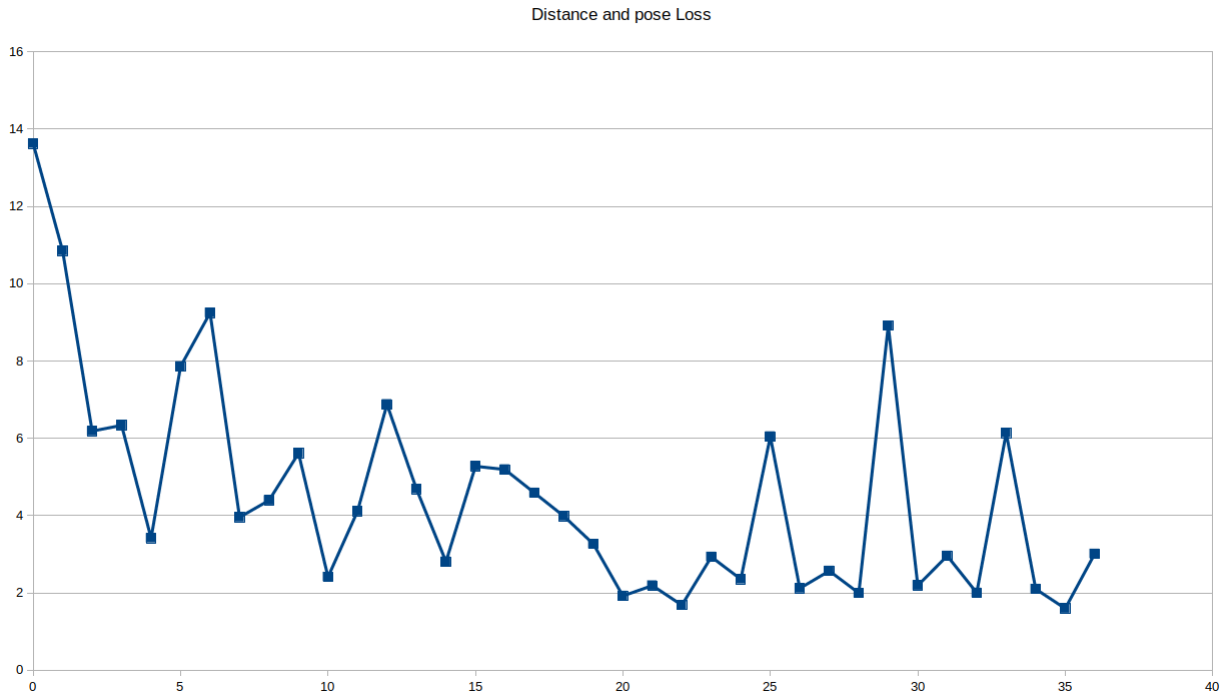

Figure 5.5: Training loss of $L_{dis} + L_{pose}$.

pervised by the grounth-ture pose.

$$L_{pose} = ||\mathbf{P} - \hat{\mathbf{P}}||_2 \tag{5.4}$$

Where $\mathbf{P}$ is the labelled pose and $\hat{\mathbf{P}}$ is the predicted pose by the network. Then the total loss function can be written as:

$$L = \alpha L_{dis} + \beta L_{pose} \tag{5.5}$$

In equation 5.5, α and β are the hyper-parameters, which are used to balance the loss function. We take 0.5 for each other in our experiment. In Fig 5.5, we plot the loss function. We can see that the function has significant fluctuation after the several epochs. Until here, we conduct that although FlowNet3D can predict the scene flow in term of 3D data. It has worst generalization to the prediction of object pose in 3D space.

5.3 Our Network Model

Our tracking network model is shown in the Fig 5.6. It consists of three stages: object optical flow estimation and segmentation, keypoint section, and iterative keypoint refinement. We

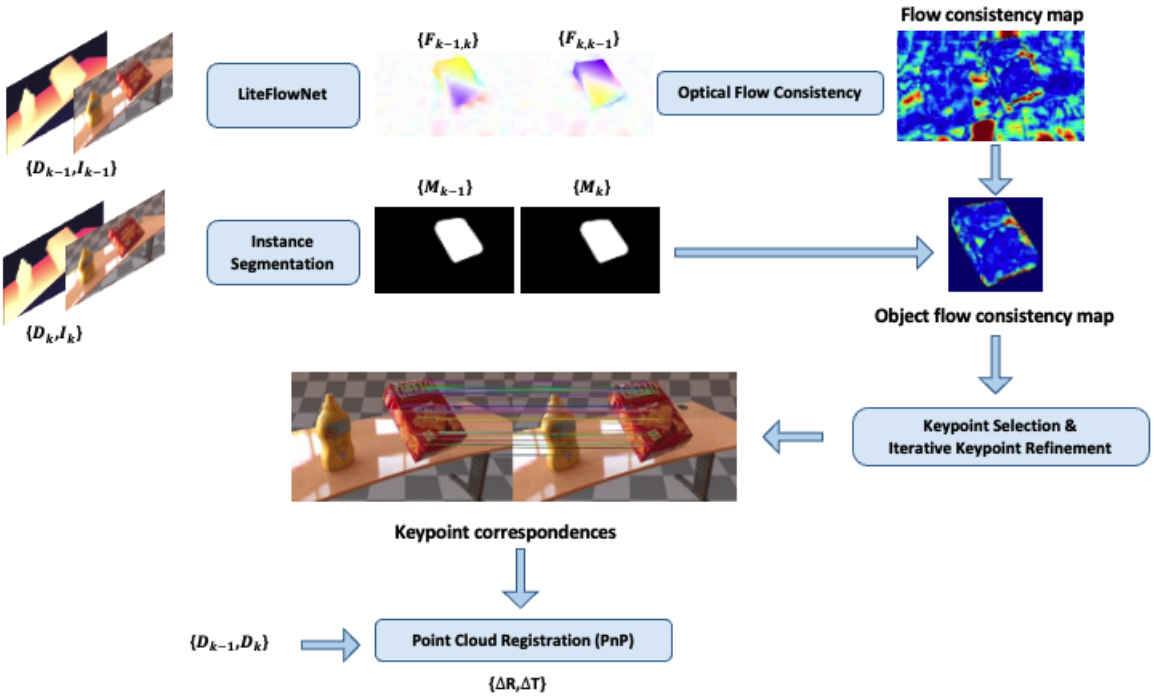

Figure 5.6: Our network model for object 6D Pose tracking

utilise the optical flow map that is extracted by a deep neural network called LiteFlowNet [56], which can find the correspondences between two image rapidly. The optical flow map records the motion information for every pixel in the image. However, we only need the motion information for tracked object. A segmentation network is used on the images (I_k and I_{k-1} in Fig. 5.6) to obtain its mask (M_k).

5.3.1 Object Optical Flow Estimation and Segmentation

In our proposed model, the object mask of object and the optical flow are computed parallel. We choose an off-the-shell CNN network named *transductive-VOS* network [116] to segment the mask of object.

Given two consecutive RGB frames I_{k-1} and I_k, the estimated flow map $\mathbf{F}_{k-1,k}$ describes the pixel motion from I_{k-1} to I_k called **forward flow**. We also calculate **backward flow** $\mathbf{F}_{k,k-1}$, which describes the pixel motion from I_k to I_{k-1}. Using these two different directional flow maps, we calculate the flow consistency $\mathbf{F}_c$ between two flows:

$$\mathbf{F}_c = w(\mathbf{F}_{k,k-1}[x]) - \mathbf{F}_{k-1,k}[x] \tag{5.6}$$

where w represents the warping operation:

$$w(\mathbf{F}_{k,k-1}[x]) = \mathbf{F}_{k,k-1}[x + \mathbf{F}_{k-1,k}[x]] \tag{5.7}$$

The $\mathbf{F}_c$ tends to be a smaller value if the forward flow and backward flow are accuracy enough. Accordingly the keypoints to be selected from the flow map have higher accuracy. In Fig 5.7, it shows the optical flow consistency for the object 009_gelatin_box from Fast-YCB dataset. The red color indicates the higher inconsistency, and blue color for lower inconsistency. The flow consistency being part of object can be cropped by the mask of object.

5.3.2 Keypoint Selection

We tend to establish the correspondences between previous and current frame for those pixels that have a high flow consistency value, meanwhile we just need to focus on the pixels of object. With the aid of object mask, we can extract the pixels that both exist in I_{k-1} and I_k. Considering the out of view or self-occlusion during the movement of object, we calculate the intersection area of two masks. Given two masks $M_{k-1} \in [0,1]^{[H,W]}$, $M_k \in [0,1]^{[H,W]}$ where 0 represents the pixel of background, 1 for the pixel of object, their intersection M is used to crop the regions from the global flow consistency map.

$$M = M_{k-1} \cap M_k, \quad M \in [0,1]^{[H,W]} \tag{5.8}$$

As shown in Fig. 5.6, the object flow consistency map is cropped by using the intersection of masks. We just select the keypoints that are located in the object. We adopt a local best-K selection scheme proposed in [117]. The region inside the object bounding box will be divided into patches of size of 5×5 and patches that don't equate to the size will be ignored. In Fig. 5.8, we show the object consistency map that has been divided into patches.

The value of patch j is calculated as:

$$C_j = min(N/L, Q_j) \tag{5.9}$$

where N is the number of valid matches calculated by the forward-backward flow consis-

Figure 5.7: Examples of optical flow consistency on object 009_gelatin_box from Fast-YCB dataset.

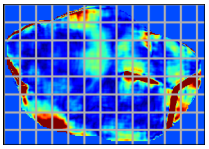

Figure 5.8: The object flow consistency map with grids

tency on entire cropped region, L is the number of divided patches, and Q_j is the number of valid matches after thresholding of forward-backward consistency in *j-th* patch. As we can see in equation 5.9, N/L represents the average distribution of keypoints on entire image. In order to let the selected keypoints remaining the geometrical information of object as much as possible, we prefer to select the them sparely. If we only choose the keypoints based on highest forward-backward consistency, it's very likely that the selected keypoints are being clustered together. Hence, we compare the keypoint numbers between average distribution and specific patch, and the smaller one is treated as the value of patch. Finally, the keypoints inside the top K valued patches are selected(local best-K). By using the local best-K selection we can increase the diversity of matched keypoints on object and avoid the potential selections on one specified region of object. Although the local best-K selection can avoid the selection from identical region, but there are still some bad quality keypoints. In below subsection, we introduce an iterative refinement scheme that can remove the outliers existed in the keypoints after local best-K selection.

5.3.3 Iterative Keypoint Refinement

The keypoint selection is based on the forward-backward consistency values. But there still exists the uncertainty in selected keypoints. For example, all the pixels share the similar prediction when the object motion is small, which make the selection very challenging. Also we found the optical flow estimation for small objects are not very stable. In our model, an **iterative keypoint refinement** scheme to improve the quality of keypoints is introduced. In this step, we calculate the rigid flow $\mathbf{F}_r$, which measures the difference between the keypoints selected by the flow consistency and the keypoints re-projected by estimated pose change $\Delta\mathbf{T}$:

$$\mathbf{F}_r = \mathbf{K}\Delta\mathbf{T}\mathbf{K}^{-1}\mathbf{D}[x] - x \tag{5.10}$$

where x is a selected keypoint, $\mathbf{D}$ is the corresponding depth, and $\mathbf{K}$ is the camera intrinsic matrix. So in the equation 5.10, **D[x]** will recover the keypoint x to 3D point. Then ,the 3D point is transformed into camera coordinate by left multiplying $\mathbf{K}^{-1}$. Then, left multiplying $\Delta\mathbf{T}$ is going to calculate the position of 3D point after movement. Then, further left multiplying $\mathbf{K}$ will project the 3D point into image frame. Therefore, the left term in equation 5.10 is actually the predicted keypoint position in the next image frame, confined by the motion of object in 3D space with pose change $\Delta\mathbf{T}$. Obviously, the position difference within keypoint in current frame and next frame is pixel motion in image, which is optical flow, but calculated with geometrical relationship. In another word, $\mathbf{F}_r$ should be consistent with the filtered flow $\mathbf{F}_c$ that are obtained after thresholding forward-backward optical flow. Their difference can be calculated below:

$$\mathbf{F}_d = |\mathbf{F}_c - \mathbf{F}_r| \tag{5.11}$$

The $\mathbf{F}_d$ is used as the criterion to check whether or not the selected keypoints encounter high uncertainty induced by the flow network. This process works as blow:

- Find a set of keypoints by forward-backward consistency.
- Calculate their rigid flow $\mathbf{F}_r$ using equation 5.10.
- Calculate $\mathbf{F}_d$ using equation 5.11.
- If $\mathbf{F}_d < \sigma_{rigid}$, the keypoint is treated as valid keypoint and re-selected for final estimation. σ_{rigid} is a pre-defined threshold for the rigid flow measurement.
- Iterate all the keypoints from forward-backward consistency. Then, we can get a new re-selected keypoint set, which forms a new 2D-2D matching.

Given the 2D-2D matching pairs between two consecutive frames, the depth image, and the keypoint positions, we recover the depth point cloud for the object. Considering the potential inaccuracy of segmentation network, the graph-cut RANSAC algorithm [118] is used to remove outliers. Afterwards, the pose change are estimated from the point cloud registration PnP via a least square approach [11].

5.4 Experiments

In this section, it describes the datasets we used for testing our network, and the metrics that use to evaluate the tracking performance. In the end, it reports the quantitative and qualitative results of our network on the selected datasets.

5.4.1 Datasets

Fast-YCB [119] dataset: the objects in this dataset are selected from the model of YCB-Video dataset, and are created from computer rendering. The motion of object in this dataset is increased moderate to fast. These photo-realistic images have more significant reflection of light compared with the dataset captured under the real-world. Object pose label, depth, mask, and optical flow are provided in the Fast-YCB dataset.

YCBInEOAT [62] dataset: this dataset contains 9 video sequences which are manipulated by a robot manipulator. In this dataset, 3 kinds of end-effector, a vacuum gripper, a Robotiq 2F-85 gripper and a Yale T42 hand are used to interact with object.

NOCS [120] dataset: this dataset contains 6 category-level objects *bottle*, *bowl*, *camera*, *can*, *laptop* and *mug*. No exact object CAD model is assumed in the dataset. It requires proposed algorithms to have the ability to recognize those unseen objects during either training or testing. Except the label of object pose, the size of object is also provided in the dataset.

There are two kinds of movement of objects in our selected datasets.

1. The camera is fixed during the tracking process only the objects are moving. Our network model is able to estimate the object pose directly. The YCBInEOAT and Fast-YCB datasets are in this category.

2. The objects are statically placed in the table-top place only the camera is moving in the front of objects. The images are captured from different poses of the camera. Our network model is able to estimate the camera pose. Then the object pose is obtained from the estimated camera pose. The NOCS dataset is in this category. In Fig 5.9, the estimated camera trajectories of *scene 1* are plotted from the of *real test* set of NOCS dataset. Although we don't have the ground-truth of the real trajectories of camera, but its accuracy can be reflected on estimation of object pose.

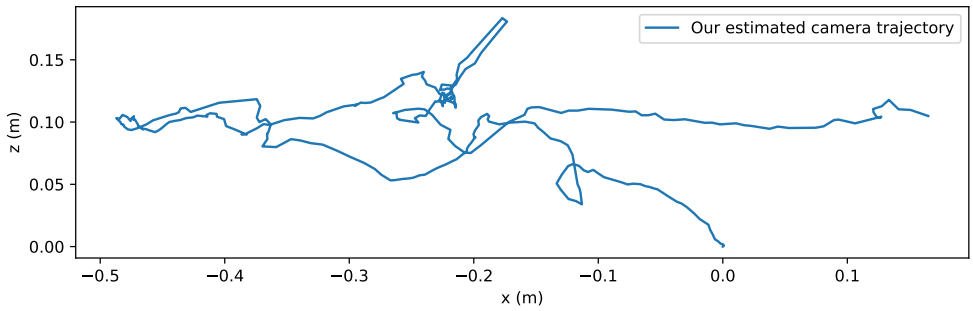

Figure 5.9: Estimated camera trajectory from NOCS dataset

5.4.2 Evaluation Metrics

We report the pose root mean square error(**RMSE**) for rotation(R_{err}) and translation(t_{err}), which computes the average precision of three Euler angles and translation vector. For the scenarios of grasping and object manipulation, the **IoU25** metric is calculated, where the overlapping volume between predicted 3D bounding box and ground-truth bounding box is higher than 25% to be counted as correct. We also report the performance of our algorithm evaluated by $5°5cm$, where the rotation error less than 5 degrees and translation error less than 5cm are treated as corrected.

The threshold for the forward-backward consistency is set to 5. The image grid of object bounding box is set to 10×10 for the Fast-YCB dataset, 30×30 for the NOCS and YCBInEOAT datasets. We set the rigid optical flow $\mathbf{F}_r$ threshold as 0.1 cm to select the best accuracy matching. During the keypoint selection from forward-backward flow consistency, we take 200 matching keypoints with the least consistent values. Then in the iterative keypoints refinement scheme, we reduce the selected keypoints to 16. The object pose change are obtained by the PnP algorithm through these 16 paired keypoints.

5.4.3 Results

Evaluation on the YCBInEOAT dataset

The table 5.1 shows the quantitative results on the YCBInEOAT dataset. The methods **Se(3)-TrackNet*** and **RGF*** with star indicate that they used the object CAD model in their pipelines. We utilised the same segmentation network and the initial pose estimation as in BundleTrack [61]. The metircs ADD and $ADD-S$ are computed and the results are treated as correct when the metric threshold is lower than 0.1 meter. The evaluation re-

sults of BundleTrack are taken from its public release, and the evaluation results of **Se(3)-TrackNet***, **RGF***, and **TEASER** are taken from their publications [61]. It is necessary to use the object model to compute the ADD and ADD-S metrics, but the original YCBInEOAT dataset doesn't provide the object models for objects: 003_cracker_box, 021_bleach_cleanser, 004_sugar_box, 005_tomato_soup_can and 006_mustard_bottle. Here, we utilised the corresponding object models from the YCB-Video dataset for the evaluation.

Methods	003_cracker_box		021_bleach_cleanser		004_sugar_box		005_tomato_soup_can		006_mustard_bottle		Mean	
	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
BundleTrack [61]	85.07	89.41	89.34	94.72	85.56	90.22	86.00	95.13	92.26	95.35	87.34	92.53
Se(3)-TrackNet* [62]	90.76	94.06	89.58	94.44	92.43	94.80	93.40	96.95	97.00	97.92	92.66	95.53
RGF* [121]	34.78	55.44	29.40	45.03	15.82	16.87	15.13	26.44	56.49	60.17	29.98	39.90
TEASER* [122]	63.24	81.35	61.83	82.45	51.91	81.42	41.36	71.61	71.92	88.53	57.91	81.17
Ours	**94.72**	**94.95**	**97.58**	**98.03**	**99.7**	**99.8**	**99.69**	**99.84**	**99.59**	**99.72**	**98.25**	**98.46**

Table 5.1: Quantitative results in the YCBInEOAT dataset by using ADD(S) metrics.

In Table 5.1, we highlight the best performance with bold number. Our method outperforms the BundleTrack for each object in the YCBInEOAT dataset, and it has more than 90% accuracy for each object. The *ADD* average accuracy is improved from 87.34% to 98.25% for *ADD*, and 92.53% to 98.46% for $ADD-S$. In the YCBInEOAT dataset, the objects are presented with small scale, which require a very precise and reliable prediction of the keypoints, and they are occluded most of the time by the manipulator. These are challenging scenes to the pose tracking tasks. The better performance from our network mainly relies on the robust keypoint selections with our proposed optical flow and segmentation networks. The Fig 5.10 shows the the matching keypoints between two consecutive frames from the YCBInEoat daset. The pose tracking performance is shown in Fig 5.11 where each row shows the estimation result for one object. It can be seen our model performs well even with the interaction of robot arm.

Evaluation on the Fast-YCB dataset

In Table 5.3, we present the quantitative results of our model compared with some other tracking frameworks. We used the same Mask-RCNN [26] as in ROFT [119] and the same pose initial estimate as in DOPE [123]. Since there is no symmetry object presented in the Fast-YCB dataset, we only utilise the *ADD* metric to evaluate the tracking performance. In the average performance, our model achieves the highest accuracy 84.28% compared with 68.10% of PoseRBPF, 76.59% of ROFT, and 72.06% of SE(3)-TrackNet. Our method leads the performance for all the objects except for the object 010_potted_meat_can for which the

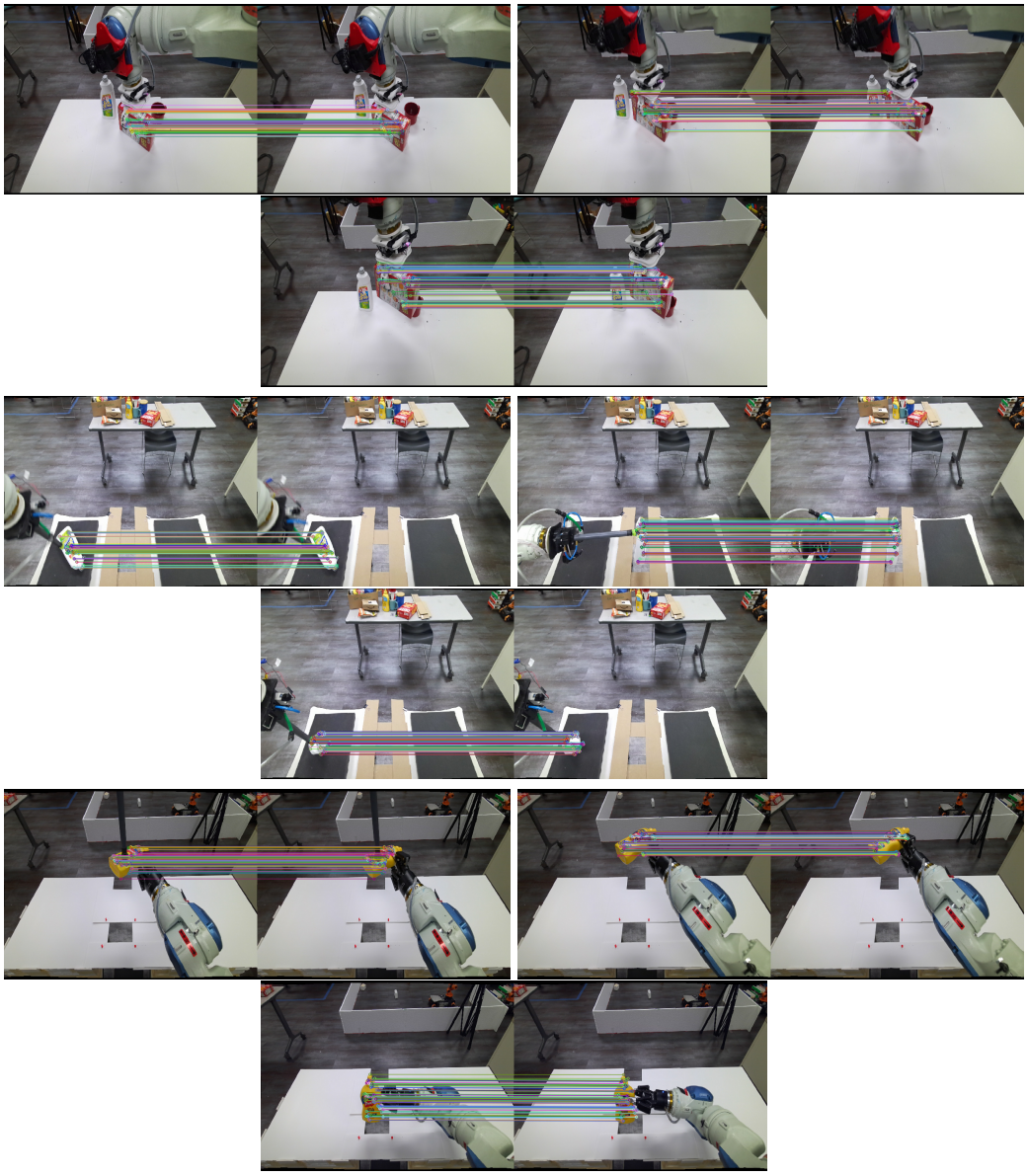

Figure 5.10: Qualitative results of keypoints selection on YCBInEoat dataset. The first two rows show the matched keypoints for the object cracker box, The second two rows show the matched keypoints for the object bleach cleanser. The last two rows show the matched keypoints for the object mustard bottle.

PoseRBPF has the best accuracy 87.29% better than 66.17% of ours. Under the rotation error R_{err}, our method has the lowest error 8.888 *deg* in Table 5.3. For the translation error t_{err}, our model significantly reduces the error to 0.355 *cm* in average, leading the second best performance of ROFT about 3 *cm*.

Figure 5.11: Qualitative results of our algorithm evaluating on the YCBInEoat dataset

	data loading	optical flow prediction	keypoint selection	pose inference	average tracking time
Fast-YCB	0.11s	0.237s	0.025s	0.002s	0.034s
YCBInEOAT	0.028s	0.083s	0.01s	0.005s	0.04s

Table 5.2: The table shows the time that spent on each section when running our system on Fast-YCB and YCBInEOAT datasets.

Evaluation on the NOCS dataset:

In Table 5.4, we report the quantitative evaluation results by our framework on the NOCS datset, where $5°5cm$, IoU25, R_{err}, and t_{err} metrics are analysed. We classify the methods for comparison as with CAD model or without CAD model. We also specify the modality of input for each method as RGB, RGD-D, Depth, and Point Cloud. For $5°5cm$ and IoU25 metrics, a higher value indicates a better performance, and a higher value indicates a worse performance for R_{err} and t_{err} metrics. We used the same segmentation network and the initial pose estimation as in BundleTrack. As we can see, our method outperforms other methods with CAD model. Especially for 6-PACK, our method largely leads the performance for the four metrics, except for the object *bowl* where 6-PACK achieves 100% on the IoU25, but our result 99.61% is very close. Also, 6-PACK has the lower rotation error for *bowl* object, which is 5.2 degree compared with 11.44 degree of ours. Our significant improvement is on the estimation of translation vector where we can see from the table that t_{err} is considerably lower than other methods. We achieve 0.013cm average translation error while the best one in other methods is 2.1cm.

Computation Time Analysis

In this section, we analyze the time consumption during the entire tracking process, shown in Table 5.2. We calculated the time of four various operations of our system, which are data loading, optical flow prediction, keypoint selection and pose inference, receptively. Among the keypoint selection, consumptions consist of the time of forward-backward flow consistency calculation, local best-K selection and rigid flow calculation. Pose inference indicates the calculation time using PnP algorithm. In the last column of Table 5.2, we analysed the average time of performing tracking on two datasets. As we can see, our system can achieve 29.4Hz on Fast-YCB dataset and 25Hz on YCBInEOAT dataset, which are close to the real time requirement of 30Hz for tracking. This tracking performance is enough for most of the pose tracking task. As the images in Fast-YCB(720×1280 pixels) have larger image size

Metric	ADD-AUC				$RMSE_t(cm)$				$RMSE_R(deg)$			
Method	DOPE [123]	PoseRBPF [60]	SE(3)-TrackNet [62]	ROFT [119]	Our	DOPE [123]	PoseRBPF [60]	SE(3)-TrackNet [62]	ROFT [119]	Our	DOPE [123]	PoseRBPF [60]	SE(3)-TrackNet [62]	ROFT [119]	Our
003_cracker_box	54.92	68.94	63.02	78.50	89.28	5.1	2.6	7.9	2.5	0.159	28.325	38.455	31.729	7.545	4.997
004_sugar_box	60.01	82.78	73.70	81.15	93.47	5.1	1.9	4.4	2.4	0.259	34.636	17.073	31.623	8.391	5.769
005_tomato_soup_can	64.14	75.93	80.82	79.00	90.31	4.7	1.1	3.3	2.9	0.471	29.556	63.811	26.282	16.571	4.288
006_mustard_bottle	57.20	82.92	74.83	73.10	74.3	8.6	2.0	11.9	3.1	0.462	36.132	18.418	35.480	13.292	14.6
009_gelatin_box	60.01	11.32	69.00	74.26	92.18	34.4	13.3	35.5	4.7	0.221	27.936	65.857	32.311	16.368	4.94
010_potted_meat_can	57.03	87.29	71.30	73.87	66.17	6.1	1.2	19.2	3.1	0.558	31.444	23.758	44.712	10.838	18.738
Average	58.83	68.10	72.06	76.59	**84.28**	15.1	5.7	17.6	3.2	**0.355**	31.491	42.979	34.155	12.675	**8.888**

Table 5.3: Quantitative results under the rotation error and translation error on Fast-YCB dataset.

Model	Methods	Input	Metrics	bottle	bowl	camera	can	laptop	mug	overall
with 3D Model	**NOCS** [120]	**RGB-D**	$5°5cm$	5.5	62.2	0.6	7.1	25.5	0.9	17.0
			IoU25	48.7	99.6	90.6	77.0	94.7	82.8	82.2
			R_{err}	25.6	4.7	33.8	16.9	8.6	31.5	20.2
			t_{err}	14.4	1.2	3.1	4.0	2.4	4.0	4.9
	6-PACK [107]	**RGB-D**	$5°5cm$	24.5	55.0	10.1	22.6	63.5	24.1	33.3
			IoU25	91.1	100.0	87.6	92.6	98.1	95.2	94.2
			R_{err}	15.6	5.2	35.7	13.9	4.7	21.3	16.0
			t_{err}	4.0	1.7	5.6	4.8	2.5	2.3	3.5
	KeypointNet [124]	**Point Cloud**	$5°5cm$	5.9	16.8	1.8	4.3	49.2	3.1	13.5
			IoU25	23.1	74.7	30.9	42.6	94.6	52.0	53.0
			R_{err}	28.5	9.8	45.2	28.8	6.5	61.2	30.0
			t_{err}	9.5	8.2	8.5	13.1	4.4	6.7	8.4
without 3D Model	**ICP** [125]	**Point Cloud**	$5°5cm$	10.1	40.3	12.6	17.2	14.8	6.2	16.9
			IoU25	29.9	79.7	53.1	40.5	50.9	27.7	47.0
			R_{err}	48.0	19.0	80.5	47.1	37.7	56.3	48.1
			t_{err}	15.7	4.7	12.2	9.4	9.2	9.2	10.5
	MaskFusion [126]	**RGB-D**	$5°5cm$	15.5	32.3	11.7	8.8	73.9	16.4	26.5
			IoU25	51.4	71.4	60.8	49.7	99.9	56.2	64.9
			R_{err}	36.7	12.3	43.0	34.9	3.4	40.6	28.5
			t_{err}	11.3	5.3	11.1	9.3	3.5	9.2	8.3
	BundleTrack [61]	**RGB-D**	$5°5cm$	86.5	99.6	85.8	99.2	99.9	53.6	87.4
			IoU	100.0	99.9	99.9	100.0	99.9	99.9	**99.9**
			R_{err}	1.6	1.7	3.0	1.5	1.5	5.2	**2.4**
			t_{err}	2.3	2.1	2.1	2.1	2.2	2.2	2.1
	CAPTRA [127]	**Depth**	$5°5cm$	79.46	79.20	0.41	64.70	94.03	55.17	74.51
			IoU25	72.11	79.64	2.50	62.47	87.20	80.70	76.42
			R_{err}	3.29	3.50	17.82	3.43	2.24	5.36	3.56
			t_{err}	2.60	1.43	35.53	5.69	1.48	0.79	2.40
	Ours	**RGB-D**	$5°5cm$	96.32	98.81	85.5	98.31	98.59	56.5	**89.5**
			IoU25	99.74	99.61	99.67	99.58	99.68	99.63	99.6
			R_{err}	4.43	11.44	3.13	4.47	1.73	5.12	5.05
			t_{err}	0.015	0.013	0.014	0.013	0.014	0.014	**0.013**

Table 5.4: Quantitative results on NOCS dataset, reported by 5^o5cm, Iou25, R_{err},t_{err} metrics.

than images in YCBInEOAT(480×640 pixels), the data loading and optical flow prediction steps of Fast-YCB occupied longer time than the time running on YCBInEOAT. However, the rest steps cost almost same time on two datasets.

5.4.4 Limitations and Future Works

In the experiment, we found that our algorithm has the limitation to handle out of plane movement of object. This resulted in the inaccuracy estimation of optical flow, further affected the performance of object pose tracking. In Fig 5.14, we showed the AUC evaluation results of all of the image sequences of 003_cracker_box and 004_sugar_box objects from Fast-YCB dataset. We can clearly see that, for some frames, our algorithm can not predict the object pose accurately , in where those frames showed the high errors for rotation and translation . As the objects in Fast-YCB dataset having more variabilities for rotation, some

of out-of-plane motions made the optical flow network hard to find the correct corresponding pixels. In the future work, we might modify our framework to overcome this limitation.

5.5 Conclusion

In this chapter, we propose an object pose tracking framework that utilizes the off-the-shelf optical flow network as a backbone network to select the effective keypoints of the object to establish a 3D-2D correspondence between the previous and current frame. Then the object pose is recovered by using a point cloud registration algorithm PnP. Our algorithm can handle the cases where the object is moving but the camera is static or the object is moving but the camera is static. Furthermore, to guarantee the robust and reliable selection of keypoints, we use an iterative keypoint refinement scheme to re-select the keypoints. The experiment results show that our framework outperforms or has a competitive performance over other tracking frameworks in selected datasets.

Figure 5.12: Qualitative results of keypoint selections on the Fast-YCB dataset. The visualizations show the selection results of 003_cracker_box, 006_mustard_bottle and 009_gelatin_box objects by each 3 image sets from top to the bottom.

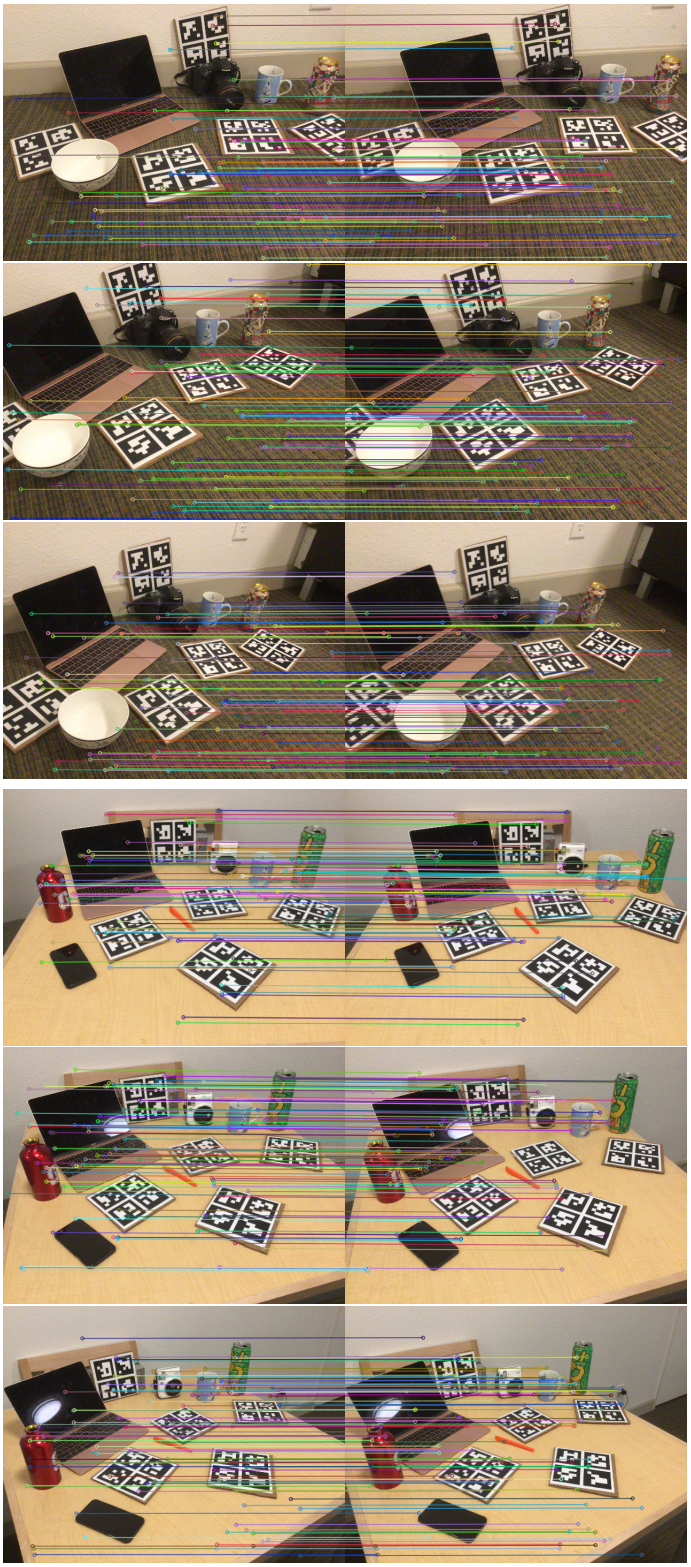

Figure 5.13: Selected keypoint matching on the NOCS dataset. The top three matching images are selected from scenes 1 of the NOCS dataset, and the bottom three matching images show the keypoint matching from scene 2 of the NOCS dataset.

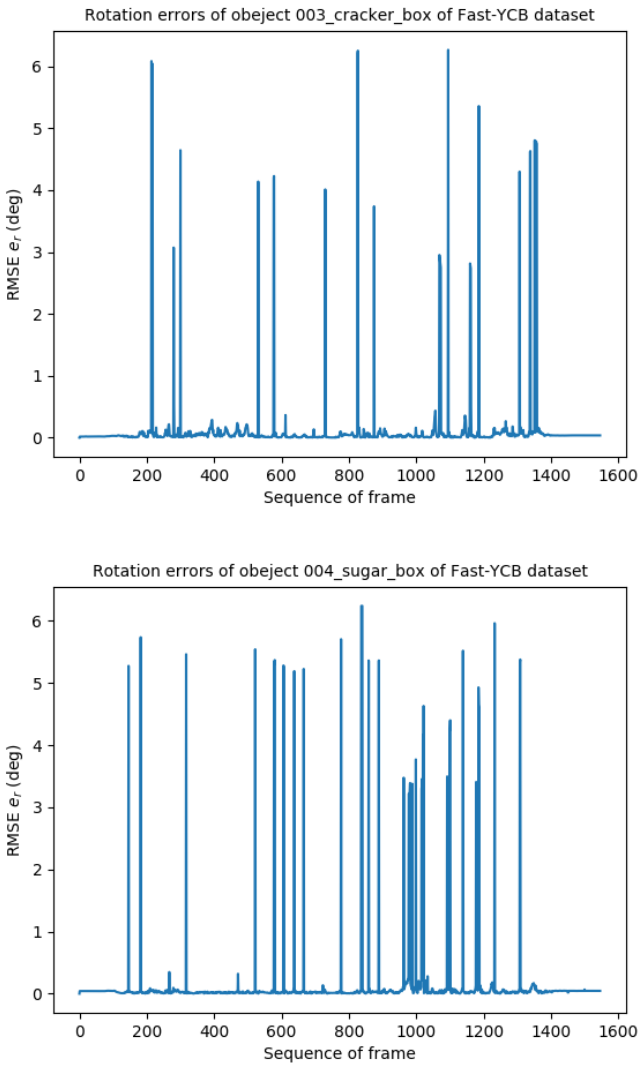

Figure 5.14: Rotation errors for the 003_cracker_box object and 004_sugar_box object, respectively, in Fast-YCB dataset. The spikes in two figures are induced by the challenging self-rotation by object, like out-of-plane rotation.

Chapter 6

Conclusions

In this chapter, we summarize the works we did in the research area of object 6D pose estimation and give the future works that we can work on to facilitate the development of this topic.

6.1 Research Summary

In this thesis, we introduced three deep neural network pipelines for object 6D pose estimation and tracking and we focused on the RGB-D data as input. We aimed to improve the accuracy of estimation, and made the estimation and/or tracking procedures more reliable when we deal with the challenging scenarios for the objects to be estimated or tracked. The first contribution in this thesis was an attention network for improving the representative of fusion features from both of the RGB and depth data. By using our proposed attention network, the simply concatenated pixel features and point features can be enhanced, so that they can improve the accuracy of object pose regression.

The second work was the graph attention network for object 6D pose estimation. Considering the local information of depth point clouds we recover from the crop images, the graph attention network took the edge information of depth information as edge features to form the fusion features so that fusion features have the combination of point features, pixel features and edge features. Then, the fusion features were processed by our proposed attention networks. For training, we constructed the graph attention network architecture in the manner of Siamese network, in order to enforce the feature space and pose space to be closed. In this way, the ambiguity of feature embeddings caused by symmetry objects could

be alleviated. Quantitative results on LineMod dataset demonstrated that pose estimation accuracy can be improved through our network.

Finally, for the third contribution, we proposed an object 6D pose tracking framework by utilizing an off the shelf optical flow network. The optical flow network took two consecutive RGB-D frames to predict the motion of objects in the image frame. Then, we selected the keypoints of object based on the predicted optical flow vector. We also proposed an iterative keypoint refinement scheme to remove the outliers among the keypoints. From the experimental results, our tracking framework demonstrated the abilities for tracking fast moving objects and occluded objects. Also, our framework is model-free, and not restricted by the shape of objects to be tracked.

6.2 Future Plans

In this section, we discussed some possible development directions of object 6D pose estimation that could benefit the research of it.

As we have noticed some limitations existed in the optical flow tracking system in Chapter 5, there are some extensions that could make the system more versatile. For tracking the fast moving objects, its velocity is usually an important factor to the application. As the depth information, mask information, optical flow information are available in our system, we can built another pipeline for the velocity prediction, similar to [119]. Furthermore, the estimation of size of object is being considered in recent proposed algorithm [128]. Therefore, for our network, we could make it as multi-tasks network to the object pose, size and velocity, simultaneously.

In below, we discuss four directions from different perspectives that can accelerate the development of object pose estimation research.

- Dataset. Although we have some datasets that were made for object 6D pose estimation. There were either presented in small scale or in simple scenes. Building a large scale dataset as well as the objects with challening poses is not easy to implement. So, some efforts have tried the computer-generated data for training the pose estimation networks. But the gap between simulation and reality still exists. Domain adaptation and domain randomization can be used to overcome the simulation to real world gap. Hence, we could make the effort for developing algorithms to bring synthetic data to

be closer to real data.

- Generalization and Standardization. At the moment, the generalization of object 6D pose estimation and tracking algorithms still have poor generalization to the applications. The estimation performance might get worse when we change the scene. For example, in real applications, lighting condition is poor. This will significantly affect the localization and estimation of objects we interest. Except the generalization ability, there is no a standardization for the object 6D pose estimation and/or tracking. Some methods require only RGB data and some RGB-D data. Some approaches need object 3D model to realize the estimation. These various requirements make the estimation pipelines hard to be deployed into industrial applications. Thus, focusing on the improvement of generalization and standardization will significantly benefit the researches on object pose estimation direction.

- Lightweight Architecture. To make the object pose estimation network being lightweight is getting more and more important. By making the network lightweight, we could reduce the usage of computational resources and improve the speed of inference. This is quite important for deploying object pose estimation algorithms into those embedded vision systems. Hence, to design lightweight as well as accurate pose estimation network should gain more research attention in the future.

- Multi-Sensor Fusion for Pose Estimation. The signals to be used for objects pose estimation in this thesis are images captured by camera. They only rely on the visual information to reason the object pose, which have limitations in some scenarios. In the future work, the algorithms proposed in this thesis can potentially be applied in scenes where multiple sensing sensor are available, like Inertial Measurement Unit(IMU), odometer, accelerator. Actually, in the robot state estimation, the above sensors are used a lot independently to measure the pose, distance and velocity. But fuse the information coming from those sensors together for a single agent or multiple-agents is quite challenging task in robotics. Each unique sensor has its own sensing frequency and different uncertainty. How do the algorithm combine those variables to get the final estimation? Kalman filter and its variants are used popular to address the different estimation uncertainty, but in complex motion, its observation model is difficult to be

constructed. So, to utilise the advantages both in probability model and deep learning model, the Attention network we proposed in Chapter 3 could be further developed. For example, we could design more branches in the network for processing various modalities data, then attention network could be used to learn the representation between observation and state models. In this manner, the entire system will be more robust than solely using deep learning network.

In real life applications, this research can combine with the robot manipulator to help human for the house-cleaning, cooking and so on. For example, in an intelligent kitchen, a cooking robot needs a precise, robust object pose estimation algorithm to accurately locate the position of utensils and estimate their pose. This kind of application requires the pose estimation generalized well on different objects because of the complex environment and objects to be recognized. Also, for a coffee-making robot, the equipped vision system must track the state of cup with real-time and high precision. The estimated state of cup or manipulator can provide the feedback to control unit to adjust the coffee making action of robot. In the state estimation of aircraft and satellite, vision-based pose estimation method is an important choice to their pose estimation as this can be done from the ground camera or from other observers in case some estimation units on the estimated aircraft and satellite are damaged. For example, a real time tracking and monitoring system on the airport can improve the operation efficiency and safety. We can imagine as the development of hardware and software, there will be more applications that can benefit from object pose estimation.

Bibliography

[1] "Https://www.tesla.com/en_GB/AI. Online, Accessed 24 November 2022."

[2] J. Kim, H. Pyo, I. Jang, J. Kang, B. Ju, and K. Ko, "Tomato Harvesting Robotic System Based on Deep-ToMaToS: Deep Learning Network Using Transformation Loss for 6D Pose Estimation of Maturity Classified Tomatoes with Side-Stem," *Computers and Electronics in Agriculture*, vol. 201, p. 107300, 2022.

[3] A. Mousavian, C. Eppner, and D. Fox, "6-DOF Graspnet: Variational Grasp Generation for Object Manipulation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2901–2910.

[4] V. Kozák, R. Sushkov, M. Kulich, and L. Přeučil, "Data-Driven Object Pose Estimation in a Practical Bin-Picking Application," *Sensors*, vol. 21, no. 18, p. 6093, 2021.

[5] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *arXiv preprint arXiv:1711.00199*, 2017.

[6] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3343–3352.

[7] "Https://opencv.org/. Online, Accessed 10 November 2022."

[8] S. Finstenvalder and W. Scheufele, "Das rückwärtseinschneiden im raum," *Zum 75-Geburtstage Verlag Herbert Wichmann*, pp. 86–100, 1937.

[9] W. Wen-tsün, "Basic Principles of Mechanical Theorem Proving in Geometries, Volume I: Part of Elementary Geometries," 1984.

[10] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[11] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[12] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems." in *Robotics: science and systems*, 2016.

[13] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D Point Cloud Based Object Maps for Household Environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[14] D. Gehrig, P. Krauthausen, L. Rybok, H. Kuehne, U. D. Hanebeck, T. Schultz, and R. Stiefelhagen, "Combined Intention, Activity, and Motion Recognition for a Humanoid Household Robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4819–4825.

[15] I. F. Mondragón, P. Campoy, C. Martinez, and M. A. Olivares-Méndez, "3D Pose Estimation Based on Planar Object Tracking for UAVs Control," in *2010 IEEE International Conference on Robotics and Automation*. Ieee, 2010, pp. 35–41.

[16] I. Dryanovski, W. Morris, and J. Xiao, "An open-source pose estimation system for micro-air vehicles," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4449–4454.

[17] S. Zakharov, I. Shugurov, and S. Ilic, "DPOD: Dense 6D Pose Object Detector in RGB Images," *arXiv preprint arXiv:1902.11020*, vol. 1, no. 2, 2019.

[18] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNet: Pixel-wise Voting Network for 6DOF Pose Estimation ," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4561–4570.

[19] C. R. Qi, O. Litany, K. He, and L. J. Guibas, "Deep Hough Voting for 3D Object Detection in Point Clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9277–9286.

[20] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "T-less: An rgb-d dataset for 6d pose estimation of texture-less objects," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 880–888.

[21] B. Drost, M. Ulrich, P. Bergmann, P. Hartinger, and C. Steger, "Introducing mvtec itodd-a dataset for 3d object recognition in industry," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 2200–2208.

[22] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective Search for Object Recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[24] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[25] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[27] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[30] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[31] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model Globally, Match Locally: Efficient and Robust 3D Object Recognition," in *2010 IEEE computer society conference on computer vision and pattern recognition*. Ieee, 2010, pp. 998–1005.

[32] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," in *Asian conference on computer vision*. Springer, 2012, pp. 548–562.

[33] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D Object Pose Estimation Using 3D Object Coordinates," in *European conference on computer vision*. Springer, 2014, pp. 536–551.

[34] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1521–1529.

[35] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "GS3D: An Efficient 3D Object Detection Dramework for Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1019–1028.

[36] M. Rad and V. 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*, 2017, pp. 3828–3836.

[37] D. Xu, D. Anguelov, and A. Jain, "PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 244–253.

[38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[39] M. Oberweger, M. Rad, and V. Lepetit, "Making deep heatmaps robust to partial occlusions for 3d object pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 119–134.

[40] G. Gao, M. Lauri, Y. Wang, X. Hu, J. Zhang, and S. Frintrop, "6D Object Pose Regression via Supervised Learning on Point Clouds," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3643–3649.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[42] K. Park, T. Patten, and M. Vincze, "Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7668–7677.

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in neural information processing systems*, vol. 27, 2014.

[44] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[45] V. A. Sindagi, Y. Zhou, and O. Tuzel, "MVX-Net: Multimodal VoxelNet for 3D Object Detection," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7276–7282.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[47] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional Block Attention Module ," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[48] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "GRAPH ATTENTION NETWORKS," *arXiv preprint arXiv:1710.10903*, 2017.

[49] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929*, 2020.

[50] A. Amini, A. S. Periyasamy, and S. Behnke, "T6D-Direct: Transformers for Multi-Object 6D Pose Direct Regression," in *DAGM German Conference on Pattern Recognition*. Springer, 2021, pp. 530–544.

[51] Amini, Arash and Periyasamy, Arul Selvam and Behnke, Sven, "YOLOPose: Transformer-Based Multi-Object 6D Pose Estimation using Keypoint Regression," *arXiv preprint arXiv:2205.02536*, 2022.

[52] L. Zou, Z. Huang, N. Gu, and G. Wang, "6D-ViT: Category-Level 6D Object Pose Estimation via Transformer-Based Instance Representation Learning," *IEEE Transactions on Image Processing*, 2022.

[53] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning Optical Flow with Convolutional Networks," *arXiv preprint arXiv:1504.06852*, 2015.

[54] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of Optical Flow Estimation with Deep Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[55] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-Net: Cnns for Optical Flow Using Pyramid, Warping, and Cost Volume," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8934–8943.

[56] T.-W. Hui, X. Tang, and C. C. Loy, "Liteflownet: A Lightweight Convolutional Neural Network for Optical Flow Estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8981–8989.

[57] Hui, Tak-Wai and Tang, Xiaoou and Loy, Chen Change, "A Lightweight Optical Flow CNN - Revisiting Data Fidelity and Regularization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 8, pp. 2555–2569, 2020.

[58] C. Choi and H. I. Christensen, "RGB-D Object Tracking: A Particle Filter Approach on GPU," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1084–1091.

[59] M. Majcher and B. Kwolek, "Deep Quaternion Pose Proposals for 6D Object Pose Tracking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 243–251.

[60] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "Poserbpf: A Rao–Blackwellized Particle Filter for 6D Object Pose Tracking," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1328–1342, 2021.

[61] B. Wen and K. Bekris, "Bundletrack: 6D Pose Tracking for Novel Objects without Instance or Category-Level 3D Models," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8067–8074.

[62] B. Wen, C. Mitash, B. Ren, and K. E. Bekris, "se(3)-TrackNet: Data-driven 6D Pose Tracking by Calibrating Image Residuals in Synthetic Domains," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 367–10 373.

[63] I. Marougkas, P. Koutras, N. Kardaris, G. Retsinas, G. Chalvatzaki, and P. Maragos, "How To Track Your Dragon: A Multi-Attentional Framework for Real-Time

RGB-D 6-DOF Object Pose Tracking," in *European Conference on Computer Vision*. Springer, 2020, pp. 682–699.

[64] F. Manhardt, W. Kehl, N. Navab, and F. Tombari, "Deep Model-Based 6D Pose Refinement in RGB," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 800–815.

[65] M. Fabbri, G. Brasó, G. Maugeri, O. Cetintas, R. Gasparini, A. Ošep, S. Calderara, L. Leal-Taixé, and R. Cucchiara, "Motsynth: How Can Synthetic Data Help Pedestrian Detection and Tracking?" in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 849–10 859.

[66] Y. Liu, Z. Wang, X. Zhou, and L. Zheng, "Synthetic Data Are as Good as the Real for Association Knowledge Learning in Multi-object Tracking," *arXiv preprint arXiv:2106.16100*, 2021.

[67] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 2011, pp. 127–136.

[68] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes ," in *2011 international conference on computer vision*. IEEE, 2011, pp. 858–865.

[69] R. Kaskman, S. Zakharov, I. Shugurov, and S. Ilic, "Homebreweddb: RGB-D Dataset for 6D Pose Estimation of 3D Objects," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[70] B. Tekin, S. N. Sinha, and P. Fua, "Real-Time Seamless Single Shot 6D Object Pose Prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292–301.

[71] M. Garon, D. Laurendeau, and J.-F. Lalonde, "A Framework for Evaluating 6-DOF Object Trackers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 582–597.

[72] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold *et al.*, "Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3364–3372.

[73] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, "Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation," in *European conference on computer vision*. Springer, 2016, pp. 205–220.

[74] T. Do, T. Pham, M. Cai, and I. Reid, "Real-Time Monocular Object Instance 6D Pose Estimation," 2019.

[75] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," *arXiv preprint arXiv:1809.10790*, 2018.

[76] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED Framework: Object Recognition and Pose Estimation for Manipulation," *The international journal of robotics research*, vol. 30, no. 10, pp. 1284–1306, 2011.

[77] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-Net 3.0: Computing Robust Vacuum Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning," in *2018 IEEE International Conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5620–5627.

[78] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[79] A. Geiger, P. Lenz, and R. Urtasun, "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[80] D. J. Tan, N. Navab, and F. Tombari, "6D Object Pose Estimation with Depth Images: A Seamless Approach for Robotic Interaction and Augmented Reality," *arXiv preprint arXiv:1709.01459*, 2017.

[81] E. Marchand, H. Uchiyama, and F. Spindler, "Pose Estimation for Augmented Reality: A Hands-On Survey," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2633–2651, 2015.

[82] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-DOF Object Pose from Semantic Keypoints," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2011–2018.

[83] S. Suwajanakorn, N. Snavely, J. Tompson, and M. Norouzi, "Discovery of Latent 3D Keypoints via End-to-end Geometric Reasoning," *arXiv preprint arXiv:1807.03146*, 2018.

[84] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, "Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation," in *European conference on computer vision*. Springer, 2016, pp. 205–220.

[85] O. H. Jafari, S. K. Mustikovela, K. Pertsch, E. Brachmann, and C. Rother, "iPose: Instance-Aware 6D Pose Estimation of Partly Occluded Objects," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 477–492.

[86] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic, "CosyPose: Consistent multi-view multi-object 6D pose estimation," in *European Conference on Computer Vision*. Springer, 2020, pp. 574–591.

[87] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[88] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual Attention Network for Image Classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3156–3164.

[89] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual Attention Network for Scene Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3146–3154.

[90] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *International conference on machine learning*. PMLR, 2015, pp. 2048–2057.

[91] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[92] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[93] A. Doumanoglou, V. Balntas, R. Kouskouridas, and T.-K. Kim, "Siamese Regression Networks with Efficient Mid-Level Feature Extraction for 3D Object Pose Estimation," *arXiv preprint arXiv:1607.02257*, 2016.

[94] P. Yin, J. Ye, G. Lin, and Q. Wu, "Graph Neural Network for 6D Object Pose Estimation," *Knowledge-Based Systems*, vol. 218, p. 106839, 2021.

[95] Y. He, H. Huang, H. Fan, Q. Chen, and J. Sun, "FFB6D: A Full Flow Bidirectional Fusion Network for 6D Pose Estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3003–3013.

[96] Y. He, Y. Wang, H. Fan, J. Sun, and Q. Chen, "FS6D: Few-Shot 6D Pose Estimation of Novel Objects," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 6814–6824.

[97] C. Song, J. Song, and Q. Huang, "Hybridpose: 6D Object Pose Estimation Under Hybrid Representations," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 431–440.

[98] S. Zhang, W. Zhao, Z. Guan, X. Peng, and J. Peng, "Keypoint-Graph-Driven Learning Framework for Object Pose Estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 1065–1073.

[99] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[100] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim, "Latent-Class Hough Forests for 3D Object Detection and Pose Estimation," in *European Conference on Computer Vision*. Springer, 2014, pp. 462–477.

[101] V. Balntas, A. Doumanoglou, C. Sahin, J. Sock, R. Kouskouridas, and T.-K. Kim, "Pose Guided RGBD Feature Learning for 3D Object Pose Estimation," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 3856–3864.

[102] S. Yan, Y. Xiong, and D. Lin, "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[103] Y. Sun, B. Cao, P. Zhu, and Q. Hu, "Drone-based RGB-Infrared Cross-Modality Vehicle Detection via Uncertainty-Aware Learning," 2020. [Online]. Available: https://arxiv.org/abs/2003.02437

[104] Z. Wang, C. Xu, and F. Gao, "Robust Trajectory Planning for Spatial-Temporal Multi-Drone Coordination in Large Scenes," *arXiv preprint arXiv:2109.08403*, 2021.

[105] "[dodecapen: Accurate 6dof tracking of a passive stylus."

[106] H. Ouyang, B. Zhang, P. Zhang, H. Yang, J. Yang, D. Chen, Q. Chen, and F. Wen, "Real-Time Neural Character Rendering with Pose-Guided Multiplane Images," *arXiv preprint arXiv:2204.11820*, 2022.

[107] C. Wang, R. Martín-Martín, D. Xu, J. Lv, C. Lu, L. Fei-Fei, S. Savarese, and Y. Zhu, "6-Pack: Category-Level 6D Pose Tracker with Anchor-Based Keypoints," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 059–10 066.

[108] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, p. 1, 2017.

[109] Y. Lin, J. Tremblay, S. Tyree, P. A. Vela, and S. Birchfield, "Keypoint-Based Category-Level Object Pose Tracking from an RGB Sequence with Uncertainty Estimation," *arXiv preprint arXiv:2205.11047*, 2022.

[110] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 683–698.

[111] N. A. Piga, F. Bottarel, C. Fantacci, G. Vezzani, U. Pattacini, and L. Natale, "Maskukf: An Instance Segmentation Aided Unscented Kalman Filter for 6F Object Pose and Velocity Tracking," *Frontiers in Robotics and AI*, vol. 8, p. 594583, 2021.

[112] E. A. Wan and R. Van Der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. Ieee, 2000, pp. 153–158.

[113] T. S. Cohen and M. Welling, "Steerable CNNs," *arXiv preprint arXiv:1612.08498*, 2016.

[114] J. Lin, H. Li, K. Chen, J. Lu, and K. Jia, "Sparse Steerable Convolutions: An Efficient Learning of SE (3)-Equivariant Features for Estimation and Tracking of Object Poses in 3D Space," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16 779–16 790, 2021.

[115] X. Liu, C. R. Qi, and L. J. Guibas, "Flownet3D: Learning Scene Flow in 3D Point Clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 529–537.

[116] Y. Zhang, Z. Wu, H. Peng, and S. Lin, "A Transductive Approach for Video Object Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6949–6958.

[117] H. Zhan, C. S. Weerasekera, J.-W. Bian, R. Garg, and I. Reid, "DF-VO: What Should Be Learnt for Visual Odometry?" *arXiv preprint arXiv:2103.00933*, 2021.

[118] D. Barath and J. Matas, "Graph-Cut RANSAC," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6733–6741.

[119] N. A. Piga, Y. Onyshchuk, G. Pasquale, U. Pattacini, and L. Natale, "ROFT: Real-Time Optical Flow-Aided 6D Object Pose and Velocity Tracking," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 159–166, 2021.

[120] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas, "Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[121] J. Issac, M. Wüthrich, C. G. Cifuentes, J. Bohg, S. Trimpe, and S. Schaal, "Depth-Based Object Tracking Using a Robust Gaussian Filter," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 608–615.

[122] H. Yang, J. Shi, and L. Carlone, "Teaser: Fast and Certifiable Point Cloud Registration," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 314–333, 2020.

[123] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," *arXiv preprint arXiv:1809.10790*, 2018.

[124] Y. You, Y. Lou, C. Li, Z. Cheng, L. Li, L. Ma, C. Lu, and W. Wang, "Keypointnet: A Large-Scale 3D Keypoint Dataset Aggregated from Numerous Human Annotations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 647–13 656.

[125] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," *arXiv preprint arXiv:1801.09847*, 2018.

[126] M. Runz, M. Buffier, and L. Agapito, "Maskfusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects," in *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2018, pp. 10–20.

[127] Y. Weng, H. Wang, Q. Zhou, Y. Qin, Y. Duan, Q. Fan, B. Chen, H. Su, and L. J. Guibas, "Captra: Category-Level Pose Tracking for Rigid and Articulated Objects from Point Clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 209–13 218.

[128] J. Lin, Z. Wei, C. Ding, and K. Jia, "Category-level 6D object pose and size estimation using self-supervised deep prior deformation networks," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*. Springer, 2022, pp. 19–34.