6D Pose Estimation and Object Recognition

Nuno José Matos Pereira

Tese para obtenção do Grau de Doutor em
Engenharia Informática
(3º ciclo de estudos)

Orientador: Prof. Doutor Luís Filipe Barbosa de Almeida Alexandre

janeiro de 2024

6D Pose Estimation and Object Recognition

Composição do Júri

Hugo Pedro Martins Carriço Proença
Professor Catedrático — Presidente
Universidade da Beira Interior

Luís Filipe de Seabra Lopes
Professor Associado — Arguente
Universidade de Aveiro

Gil Manuel Magalhães de Andrade Gonçalves
Professor Auxiliar — Arguente
Faculdade de Engenharia da Universidade do Porto

João Carlos Raposo Neves
Professor Auxiliar — Arguente
Universidade da Beira Interior

Luís Filipe Barbosa de Almeida Alexandre
Professor Catedrático — Orientador
Universidade da Beira Interior

Provas realizadas a 17 janeiro 2024 com início às 14:30 horas.

Declaração de Integridade

Eu, Nuno José Matos Pereira, que abaixo assino, estudante com o número de inscrição D2058 do 3º Ciclo de Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Assinado por: **Nuno José Matos Pereira**
Num. de Identificação: 13897941
Data: 2024.01.23 08:28:37+00'00'

CHAVE MÓVEL

Universidade da Beira Interior, Covilhã 21/01/2024

Acknowledgements

I would like to express my deepest gratitude to the individuals and organizations who have been instrumental in the completion of this thesis.

First and foremost, I want to thank my parents for their unwavering support, encouragement, and love throughout my journey. I am truly grateful for their understanding and patience during the challenging times of this research work.

I extend my heartfelt appreciation to the members of the Soft Computing and Image Analysis Lab (SOCIA-LAB). Their collaboration, expertise, and the enjoyable working environment.

I am indebted to my supervisor, Professor Luís Filipe Barbosa de Almeida Alexandre, for his exceptional guidance, unwavering support, and invaluable mentorship. His expertise, research insights, and dedication have been invaluable throughout the years. I am truly grateful for his patience, trust, and encouragement that have pushed me to strive for the best research I could do.

Additionally, I would like to acknowledge the support and financial assistance provided by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP, and partially supported by project 026653 (POCI-01-0247-FEDER-026653) INDTECH 4.0 – New technologies for smart manufacturing, cofinanced by the Portugal 2020 Program (PT 2020), Compete 2020 Program and the European Union through the European Regional Development Fund (ERDF).

Resumo

A estimação da postura 6D de objetos é uma tarefa de visão computacional cujo objetivo é estimar os 3 graus de liberdade da posição do objeto (vetor de translação) e os outros 3 graus de liberdade para a orientação do objeto (matriz de rotação). A estimação da postura 6D de objetos é um problema difícil de resolver devido à possível variabilidade de iluminação, desordem, ocultação, diferentes formas, tamanhos, texturas e semelhanças entre os objetos na cena. No entanto, métodos de estimação de postura 6D são usados em múltiplos contextos, como em realidade aumentada, por exemplo, onde objetos mal colocados no mundo real podem prejudicar a experiência da realidade aumentada. Outro exemplo de aplicação é o uso de realidade aumentada na indústria para treinar novos trabalhadores e competentes, onde objetos virtuais precisam de ser colocados nas posições corretas para se assemelharem aos objetos reais ou simular a sua colocação nas posições corretas. No contexto da Indústria 4.0, os sistemas robóticos requerem adaptação para lidar com tarefas de pegar e colocar não restritas, interação e colaboração humano-robô e movimento autónomo. Esses ambientes e tarefas dependem de métodos que realizam deteção, localização, segmentação e estimação de postura 6D de objetos. Para ter uma manipulação robótica precisa, sem necessidade de utilizar contraformas, é preciso uma compreensão e perceção do ambiente envolvente. Para conseguirmos estes objetivos são necessários métodos precisos de deteção e estimação da postura 6D de objetos.

Esta tese apresenta métodos que foram desenvolvidos para resolver o problema de estimação de postura 6D, bem como a implementação dos fluxos de trabalho propostos para o mundo real. Para usá-los, foi necessário capturar e anotar um conjunto de dados para treinar e testar estes métodos no mundo real. Foi também necessário desenvolver algumas interfaces de controlo para conseguir implementar os fluxos de trabalho propostos num robô UR3.

O método MaskedFusion, proposto por nós, alcança uma estimação da postura com erro abaixo de $6mm$ no conjunto de dados LineMOD e uma avaliação de AUC de 93,3% no conjunto de dados YCB-Video. Apesar do tempo de treino mais longo que métodos anteriores, o MaskedFusion demonstra um tempo de inferência baixo, tornando-o adequado para aplicações em tempo real. Foi também realizado um estudo sobre a eficácia da implementação de diferentes espaços de cores e algoritmos de segmentação para aperfeiçoar a precisão dos métodos de estimação de postura 6D.

O método MPF6D, também proposto nesta tese, supera todas as outras abordagens, alcançando uma precisão notável de 99,7% no conjunto de dados LineMOD e 98,06% no conjunto de dados YCB-Video, demonstrando o seu potencial para estimação de postura

6D de alta precisão. A tese apresenta ainda métodos de pega de objetos com precisão elevada. A primeira abordagem, que compreende captura de dados, deteção de objetos, estimação de postura 6D, deteção da pega, planeamento da movimentação do robô e sua execução, alcança uma taxa de sucesso de 90% em testes em ambiente não controlado. O uso de um conjunto de dados diversificado com condições de iluminação variáveis é crucial para um desempenho preciso em cenários do mundo real. Além disso, um método alternativo demonstra uma pega precisa de objetos sem depender da estimação de postura 6D, oferecendo uma execução mais rápida e exigindo menor poder computacional. Com uma precisão notável de 96% e um tempo médio de execução de 5,59 segundos num computador portátil que não dispõe de uma GPU da NVIDIA, este método demonstra eficiência e aplicabilidade ao realizar tarefas de pegar e colocar objetos num ambiente não restritivo, usando um robô UR3.

Concluímos que as técnicas por nós propostas permitiram avançar o estado da arte na área da estimação de postura 6D de objetos, contribuindo assim para a criação de sistemas mais responsivos e com melhor comportamento, sempre que seja necessário obter este tipo de informação a partir de dados reais.

Palavras-chave

Inteligência Artificial, Postura de Objectos, *Deep Learning*, Redes Neuronais Artificiais, Redes Neuronais de convolução, Segmentação de imagens, *Semantic Segmentation*, Robótica, *Pick-and-place*, *Object Grasping*.

Resumo Alargado

A estimação de postura 6D é um problema de visão computacional que tem muitas aplicações importantes em diversas áreas da indústria, como por exemplo, a robótica e a realidade aumentada.

No contexto da robótica, uma postura precisa 6D do objeto é crucial para muitas tarefas, como a pega de objetos e manipulação/movimentação robótica. Por exemplo, numa fábrica, um braço robótico pode precisar de pegar num objeto de uma caixa e colocá-lo numa posição especifica numa linha de montagem. Para isso ser possível, o robô tem de ser capaz de estimar com precisão a postura 6D do objeto para o agarrar na orientação correta e colocá-lo na posição pretendida. Da mesma forma, num armazém ou centro de distribuição, robôs podem precisar pegar e colocar objetos de várias formas e tamanhos em locais diferentes. Uma estimativa precisa da postura 6D é necessária para garantir que o robô realiza essas tarefas de maneira confiável e eficiente.

No contexto da realidade aumentada, a estimação da postura 6D é usada para sobrepor objetos virtuais ao mundo real de forma que pareça realista. Por exemplo, pode ser preciso colocar objetos virtuais no ambiente do mundo real de forma que pareça que estes estão realmente no local. Uma estimação precisa da postura 6D é necessária para garantir que os objetos virtuais sejam colocados na posição e orientação corretas em relação ao ambiente do mundo real.

Em geral, a estimação da postura 6D é um problema importante em visão computacional com muitas aplicações práticas. Apesar dos desafios envolvidos em enfrentar esse problema, existem muitas abordagens e diversas técnicas de inteligência artificial que foram desenvolvidas para tentar resolver o problema.

Alcançar uma estimação robusta da postura 6D de objetos no mundo real é um desafio ainda em aberto, devido às variadas formas de objetos, tipos de materiais e de cores combinadas com múltiplas condições de iluminação, tornando esta área do campo de visão computacional difícil de implementar no mundo real. Para enfrentar esse desafio, propusemos 4 novos métodos.

Inicialmente, tentámos melhorar a arquitetura DenseFusion sem considerar a fase de refinamento de postura do objeto. Concluímos que 30 épocas são suficientes para treinar o DenseFusion original sem o refinamento, sendo possível obter bons resultados em termos de estimação de postura, usando o mínimo de tempo possível para treinar o método.

A fase de refinamento de estimação ajuda o método a alcançar maior precisão com a desvantagem de necessitar de mais tempo para treinar. Demonstramos também, que o uso de descritores globais em vez do algoritmo PointNet, que serve para extrair características das nuvens de pontos, poderá ser uma vantagem, visto ser necessário menos 19% de tempo para treinar este tipo de arquitetura, com um aumento de $1mm$ no erro da postura. No mundo real, pode ser vantajoso quando for necessário adicionar um novo objeto, dado que o método precisará ser ajustado ou re-treinado, envolvendo altos custos temporais e de computação.

O MaskedFusion, proposto por nós, melhorou os métodos anteriores nesta área utilizando máscaras de objetos, estimadas com segmentação semântica, para identificar e localizar o objeto na imagem RGB. Com este método foi alcançado um erro abaixo de 6 mm no LineMOD com apenas 100 épocas de treino. No YCB-Video, que é um conjunto de dados mais desafiante, este obteve uma AUC de 93,3% e 97,1% na métrica $< 2cm$. As máscaras obtidas pelo algoritmo de segmentação são usadas para remover dados não relevantes da entrada da rede neuronal responsável pela estimação da postura 6D. As máscaras dos objetos servem também como dados adicionais para extrair características da forma do objeto. O MaskedFusion tem um baixo tempo de inferência em comparação com os métodos anteriores, mas aumentou o tempo de treino. O aumento do tempo de treino pode por vezes ser ignorado, desde que o MaskedFusion ainda obtenha resultados melhores que os métodos anteriores. O uso de máscaras pode ser visto como um tipo de mecanismo de atenção, forçando a rede a processar apenas a região destacada pela máscara.

Após a criação do MaskedFusion, apresentámos várias experiências realizadas para analisar ainda mais dois fatores importantes que influenciam o problema de estimativa de postura 6D: a qualidade das máscaras de segmentação para o MaskedFusion e o espaço de cores usado para representar as imagens capturadas para o MaskedFusion e o DenseFusion. Com estas experiências concluímos que, usar diferentes espaços de cores pode melhorar a precisão da postura estimada. Para o MaskedFusion, que usa máscaras de segmentação semântica dos objetos para remover o fundo e informações não relacionadas com o objeto, usar métodos de segmentação semântica que obtenham melhor precisão das máscaras e aplicar filtros de pós-processamento para suavizar as máscaras obtidas, melhora a precisão da postura 6D dos objetos. Em termos do uso de diferentes espaços de cores para a estimação da postura 6D, aprendemos que, se estimarmos posturas de objetos coloridos, o espaço de cores HSV pode melhorar a precisão nesses objetos dado que as características da cor podem ser mais relevantes para estas redes neuronais. Para objetos com texturas, por exemplo, objetos embalados em caixas com rótulos e desenhos, o espaço de cores HSV não teve um desempenho tão satisfatório como noutros casos, concluindo que, para esses tipos de objetos, devemos usar o espaço de cores Gray ou RGB. No geral, usando o espaço de cores HSV, melhoramos o MaskedFusion em 1,1% e o DenseFu-

sion em 0,3% no conjunto de dados LineMOD e 0,3% para o MaskedFusion e 0,4% para o DenseFusion no conjunto de dados YCB-Video. Estas são pequenas melhorias, sendo difícil obter ganhos significativos, já que estes métodos já têm uma elevada precisão.

Depois dos diversos estudos elaborados, propusemos um novo método que consiste numa única rede neuronal que é capaz de realizar a inferência completa, desde os dados até à estimação da postura 6D. O método pode ser usado em tempo real, levando apenas 0,12 segundos para obter uma estimação precisa da postura 6D de um objeto conhecido presente em cena. O método teve o melhor desempenho geral nos dois conjuntos de dados utilizados (LineMOD e YCB-Video). No conjunto de dados LineMOD, alcançámos uma precisão de 99,7%, aumentando a precisão em 0,3% em relação ao segundo melhor método PVN3D. No conjunto de dados YCB-Video, alcançámos uma AUC de 98,06% na métrica ADD-S, o que é 1,96% superior ao PVN3D. Realizámos estudos de ablação para esclarecer o impacto dos três principais componentes da arquitetura nas taxas de erro geral, e concluímos que a remoção desses componentes leva a um aumento de erro semelhante nos dois conjuntos de dados usados, indicando que os benefícios não são dependentes dos dados utilizados. Ao longo desta tese, propusemos novas abordagens para este desafio em aberto, melhorámos esta área com os métodos desenvolvidos e conseguimos que os nossos métodos obtivessem bons tempos de inferência.

Demonstrou-se também que a utilização de um método robusto e preciso de estimação de postura 6D permite uma elevada precisão na tarefa de pegar e manobrar os objetos. A sequência proposta, envolve a captura de dados, deteção de objetos ou segmentação de objetos, estimativa de posição 6D, deteção da pega de objetos, planeamento da movimentação do robô e execução dos movimentos planeados. Esta sequência obteve uma taxa de sucesso de 90% nos testes executados em ambientes não controlados. A utilização do nosso sistema de captura de dados para criar diferentes condições de luz no conjunto de dados contribuiu para uma elevada precisão em diferentes condições de luz existentes no mundo real. Além disso, verificámos que o conjunto de dados usado deveria ter melhores condições de luz, já que ao testar com outros tipos de condições de luz foi possível observar algumas falhas nos métodos treinados neste conjunto de dados. Para usar estes métodos em cenários reais, é necessário um grande conjunto de dados com múltiplas condições de luz e fundos para se conseguir obter os melhores resultados.

Numa segunda abordagem que propusemos, conseguimos com elevada precisão pegar nos objetos sem termos de usar a estimação da postura 6D dos mesmos. Esse método é mais rápido e requer menos poder computacional para executar uma pega e movimentação eficiente de objetos no mundo real. Este método alcançou 96% de precisão, usando apenas, em média, 5,59 segundos para fornecer as coordenadas necessárias para executar a pega e a movimentação de um objeto conhecido. Estes valores foram obtidos usando

um computador portátil com baixa capacidade de processamento que não tinha GPU e usando um robô UR3. Este método não precisou de estimação de postura 6D devido à pega dos objetos em questão ser sempre executada pelo topo dos mesmos, ou seja, 90 graus perpendicular à superfície.

Concluímos que as técnicas por nós propostas permitiram avançar o estado da arte na área da estimação de postura 6D de objetos, contribuindo assim para a criação de sistemas mais responsivos e com melhor comportamento, sempre que seja necessário obter este tipo de informação a partir de dados reais.

Abstract

6D pose estimation is a computer vision task where the objective is to estimate the 3 degrees of freedom of the object's position (translation vector) and the other 3 degrees of freedom for the object's orientation (rotation matrix). 6D pose estimation is a hard problem to tackle due to the possible scene cluttering, illumination variability, object truncations, and different shapes, sizes, textures, and similarities between objects. However, 6D pose estimation methods are used in multiple contexts like augmented reality, for example, where badly placed objects into the real-world can break the experience of augmented reality. Another application example is the use of augmented reality in the industry to train new and competent workers where virtual objects need to be placed in the correct positions to look like real objects or simulate their placement in the correct positions. In the context of Industry 4.0, robotic systems require adaptation to handle unconstrained pick-and-place tasks, human-robot interaction and collaboration, and autonomous robot movement. These environments and tasks are dependent on methods that perform object detection, object localization, object segmentation, and object pose estimation. To have accurate robotic manipulation, unconstrained pick-and-place, and scene understanding, accurate object detection and 6D pose estimation methods are needed.

This thesis presents methods that were developed to tackle the 6D pose estimation problem as-well as the implementations of proposed pipelines in the real-world. To use the proposed pipelines in the real-world a data set needed to be capture and annotated to train and test the methods. Some controlling robot routines and interfaces were developed in order to be able to control a UR3 robot in the pipelines.

The MaskedFusion method, proposed by us, achieves pose estimation accuracy below $6mm$ in the LineMOD dataset and an AUC score of 93.3% in the challenging YCB-Video dataset. Despite longer training time, MaskedFusion demonstrates low inference time, making it suitable for real-time applications. A study was performed about the effectiveness of employing different color spaces and improved segmentation algorithms to enhance the accuracy of 6D pose estimation methods.

Moreover, the proposed MPF6D outperforms other approaches, achieving remarkable accuracy of 99.7% in the LineMOD dataset and 98.06% in the YCB-Video dataset, showcasing its potential for high-precision 6D pose estimation. Additionally, the thesis presents object grasping methods with exceptional accuracy. The first approach, comprising data capture, object detection, 6D pose estimation, grasping detection, robot planning, and motion execution, achieves a 90% success rate in non-controlled environment tests. Leveraging a diverse dataset with varying light conditions proves critical for accurate perfor-

mance in real-world scenarios. Furthermore, an alternative method demonstrates accurate object grasping without relying on 6D pose estimation, offering faster execution and requiring less computational power. With a remarkable 96% accuracy and an average execution time of 5.59 seconds on a laptop without an NVIDIA GPU, this method demonstrates efficiency and practicality performing unconstrained pick-and-place tasks using a UR3 robot.

Keywords

Artificial Intelligence, 6D Pose Estimation, Deep Learning, Artificial Neural Networks, Convolution Neural Networks, Image Segmentation, Semantic Segmentation, Robotics, Pick-and-place, Object Grasping.

Contents

List of Figures

List of Tables

Acronyms

ADD	Average Distance of Model Points
ADD-S	Average Closest Point Distance
APC	Amazon Picking Challenge
AUC	Area Under the Curve
CAE	Convolution Autoencoder
CNN	Convolution Neural Networks
CPU	Central Processing Unit
ESF	Ensemble of Shape Functions
FCNN	Fully Convolutional Neural Network
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
PCL	Point Cloud Library
PnP	Perspective-n-Point
RANSAC	Random Sample Consensus
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Liner Unit
RNN	Recurrent Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
SS	Semantic Segmentation
TSDF	Truncated Signed Distance Function
UR	Universal Robot
VFH	Viewpoint Feature Histogram

Chapter 1

Introduction

This thesis regards the problem of object 6D pose estimation using RGB-D data, and makes three major contributions: 1) a new method that improved the state-of-the-art and that could be used in real-time; 2) a study of the impact of the color spaces in the 6D pose estimation; 3) a new method that leverages pyramid neural network architecture and multiple heads that can estimate the 6D pose of objects in a single forward pass with high accuracy and in real-time. Beside these contributions we also implemented the methods in a real robot to execute the objective task, unconstrained pick-and-place. To be able to use these methods in the real-world we needed to develop a data set and create an API/Library to control the robot and calibrate it with the view point of and Intel RealSense RGB-D camera. In the end we also applied quantization methods and optimized the developed pipelines to enable us to use these methods in lower end devices with high accuracy and using as less time as possible. The focus and scope of the thesis are further described in this chapter.

1.1 Motivation of the Research

Object detection and pose estimation are very important tasks for robotic manipulation, scene understanding, autonomous drive, and augmented reality. As of recent, it has been the target of much research. Specially, with the increasing automation and the need for robots that can work in non-restricted environments, the capacity to understand the scene in 3 dimensions is becoming a must. One of the main tasks involving robots is pick-and-place (grasping) of objects. Performing gasping in a non-restricted or/and cluttered environment, such as bin picking, is a tough problem to tackle. The new unconstrained environments in Industry 4.0 require adaptation of the robotic systems to handle these unconstrained pick-and-place tasks, human-robot interaction and collaboration, and autonomous robot movement. These environments and tasks are dependent on methods that do object detection and object pose estimation. To achieve accurate robotic manipulation, including unconstrained pick-and-place, and scene understanding, it needs accurate object detection and pose estimation methods. These methods are also used in other contexts like augmented reality. For example, poor placement of objects in the real-world could break the experience of augmented reality. Another application example in Industry 4.0 is the use of augmented reality to train new, competent workers in the placement of virtual objects and the correct positions for the objects in order to appear as real objects. Also

Figure 1.1: 6D Pose Estimation Example. The red dots represent the object keypoints of the estimated 6D pose projected onto the RGB image.

in the field of augmented reality, 6D pose estimation is used to measure objects, rooms, and places. Aside from these main motivations, there have been an increasing number of applications of pose estimation developed over the last few years. Autonomous vehicles recently started using the technology to recognize orientations of the road signs, traffic flow, and moving obstacles.

1.2 Problem Statement and Objectives

6D pose estimation is a computer vision task where the objective is to estimate the 3 degrees of freedom of the object's position and the other 3 degrees of freedom for the object's orientation. In Figure 1.1, a 6D pose estimation example is presented where the red dots represent the keypoints of the object projected into the RGB image. It is possible to see that some red dots are not well aligned with the object. That means there is some error between the estimation and the real object position and rotation.

6D pose estimation is a tough problem to tackle due to the possible scene cluttering, lightning problems, object truncations, background clutter, and inadequate information. However, the biggest concern might be the millions of different objects that exist in the

(a) Fully Metallic Object. (b) Meshed Office Garbage Bin.

Figure 1.2: Example of objects that are hard to capture with a RGB-D camera.

real-world with different shapes, sizes, textures, and similarities between them. Some objects, even in the same category/label, can have different shapes and sizes. Such as a simple a glass of water or a light bulb. Obtaining the data to retrieve the 6D pose is also a problem. Many of the most accurate methods require the use of RGB-D data. This type of data can be hard to obtain for certain kinds of objects. For example, fully metallic (Figure 1.2 (a)) objects and meshed office garbage bins (Figure 1.2 (b)).

For some objects, obtaining the depth data is a hard task since the most used RGB-D cameras use the projection system of patterns. These patterns refract on objects that are made of polished metal, or in the meshed office garbage bin example, the pattern might match the holes of the mesh and do not give us a correct depth image. Thus meaning that the input data captured from the real-world will have problems, noise, or miss calculated depth maps.

1.3 List of Publications

The following list of publications represents the culmination of the research efforts undertaken during the course of the doctoral program. These publications showcase the research conducted and highlight the significant contributions made to the academic community. In the subsequent sections of this document, each publication will be further described, providing insight into the research questions addressed, the methodologies employed, and the key findings obtained.

Direct relation with the mentioned work:

- [LPA19] V. Lopes, N. Pereira, and L. A. Alexandre, “Robot workspace monitoring

using a blockchain-based 3D vision approach," in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2812–2820, 2019.

- [PA19] N. Pereira and L. A. Alexandre, "Using pcl gobal descriptors in a densefusion architecture," in 25th Portuguese Conference on Pattern Recognition (RECPAD), 2019.
- [PA20b] N. Pereira and L. A. Alexandre, "MaskedFusion: Mask-based 6D object pose estimation," in 19th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 71–78, 2020.
- [PA20a] N. Pereira and L. A. Alexandre, "Exploring the impact of color space in 6D object pose estimation," in 26th Portuguese Conference on Pattern Recognition (RECPAD), 2020.
- [PA21] N.Pereira and L.A.Alexandre,"Impact of segmentation and color spaces in 6D pose estimation," in IEEE International Conference on Autonomous Robot Systems and Competitions, 2021.
- [PA22] N. Pereira and L. A. Alexandre, "An efficient pick-and-place pipeline based on quantized segmentation," in International Conference on Machine Learning, Control, and Robotics (MLCR 2022), October 2022.
- [PA23] N. Pereira and L. A. Alexandre, "MPF6D: Masked Pyramid Fusion 6D Pose Estimation," Pattern Analysis and Applications, 2023.

Collaborations during the development time of the presented work:

- [LAP22] V. Lopes, L. A. Alexandre, and N. Pereira, "Controlling robots using image analysis and a consortium blockchain," in 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 637–642, 2022.
- [LFPA21] V. Lopes, M. Fernandes, N. Pereira, and L. Alexandre, "A time-segmented consortium blockchain for robotic event registration," in The 3rd International Conference on Blockchain Technology (ICBCT 2021), 2021.

1.4 Document Organization

The present document is organized in several chapters that cover different aspects of the research work. After this Introduction the document has an Overview chapter, which presents a general introduction to the technologies and methods used in the presented work. This chapter aims to provide the reader with a context for understanding the main contributions and results of the research.

The third chapter, Related Work, presents a review of the literature on the topic of 6D pose estimation. This chapter aims to identify the current state of the art and the gaps in the literature that the present research aims to fill and it serves as a foundation for the subsequent chapters.

The fourth chapter, 6D Pose Estimation Proposals, presents the main contributions in terms of 6D pose estimation. This chapter describes the proposed algorithms and methods for estimating the 6D pose of objects in a scene. The chapter also includes a comprehensive evaluation of the proposed methods, including experimental results and comparisons with related methods.

The fifth chapter, Robot Grasping Approaches, presents the proposed methods for robot grasping, which build on the 6D pose estimation proposals presented in the previous chapter. This chapter describes the methods for planning and executing robot grasping actions based on the estimated 6D pose of the target object. The chapter also includes a thorough evaluation of the proposed grasping approaches, including experimental results.

Finally, the Conclusion and Further Research, summarizes the main contributions of the research and provides a critical assessment of the achievements and limitations of the proposed methods. This chapter also discusses the implications of the research for the broader field of robotics and suggests future directions for research in this area. This chapter also discusses some of the challenges and opportunities for advancing the research in 6D pose estimation and robot grasping.

Chapter 2

Overview

In this chapter, we will introduce the reader into some techniques and concepts that are required by our methods. We will present 3D concepts like point clouds, and present some neural network architectures. The most relevant data sets and evaluation metrics are also presented in this chapter.

2.1 Point Cloud

A point cloud is a set of data points in space that represents a scene, a 3D shape, or an object. Each point in space is represented by a set of Cartesian coordinates ($[x, y, z]$).

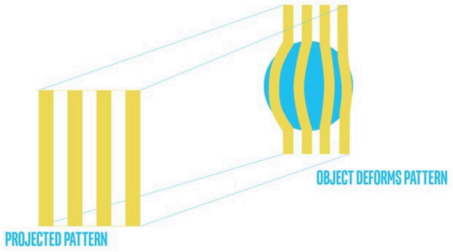
Images from conventional color digital cameras have a 2D grid of pixels, where each pixel contains three values. Typically, we refer to these values as RGB (Red, Green, Blue). The photos that we are all familiar with are a 2D grid of thousands to millions of pixels, depending on which sensor is being used. On the other hand, a depth camera uses pixels that have a unique numerical value that represents the depth distance from the camera. Some depth cameras feature both an RGB and a depth system, allowing them to produce RGB-D pixels, which are pixels with all four values.

There are numerous ways to determine depth, each having unique advantages and disadvantages, as well as, ideal conditions for use. Often what you're aiming to develop will likely affect the cameras or sensors that are used to capture this data. Depending on the range, the required precision, and the environment, some capture methods may not be an option.

There are three main types of depth capture methods that cameras or sensors use: structured/coded light, stereo depth, and LiDAR.

Structured/coded light cameras rely on emitting light onto the scene, often infrared light. It is possible for the projected light to be patterned visually, over-time, or using both. Since the projected pattern is known, the depth information comes from how the camera's sensor interprets the deformed pattern in the projected scene. For instance, if a

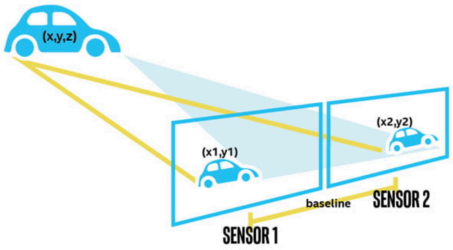
(a) Intel RealSense SR305



(b) Structured/coded light example

Figure 2.1: Figure (a) shows the Intel RealSense camera that uses coded light to capture depth data. Figure (b) displays an example of coded light. Images from [Cor]

(a) Intel RealSense D435

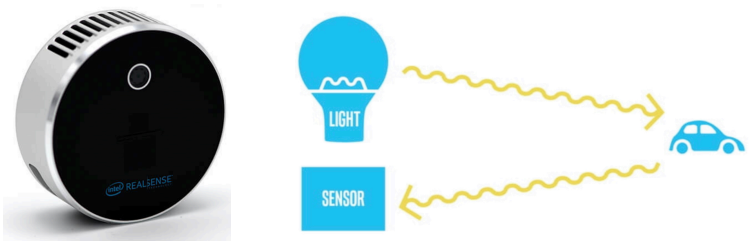


(b) Stereo Depth example

Figure 2.2: Figure (a) shows the Intel RealSense camera that uses stereo depth method to capture RGB-D data. Figure (b) displays an example of stereo depth. Images from [Cor]

series of stripes were projected onto a ball as the pattern, it would deform and bend specifically around the ball surface. The pattern projected onto the scene will also change if the ball is closer or further away from the camera. Every pixel's distance from the camera can be determined using the discrepancy between a predicted image and the actual image captured by the camera. These camera types perform best indoors at relatively close ranges since this technology depends on precisely seeing a projected pattern. This depth system is vulnerable to outside noise from other cameras or infrared-emitting gadgets.

To increase the accuracy in the captured data, stereo depth cameras frequently shine infrared light onto a scene, although, unlike structured/coded light cameras, stereo cameras can measure depth with any light. Two sensors are located close to one another in stereo depth cameras. The two images from these two sensors are compared. These comparisons provide depth information because the distance between the sensors is known. Similar to how our two eyes work together to perceive depth, stereo cameras also work in this manner. While an item in the far distance will appear to move very little, objects closer to us will appear to move dramatically from eye to eye (or sensor to sensor). Stereo cameras may be used outdoors and in most lighting conditions since they can measure depth using any visual cue. With the addition of an infrared projector, the camera can still detect depth features in dim lighting. There are no restrictions on the number of these depth cameras that can be used in a given area, and they do not interfere with other cameras the



(a) Intel RealSense L515

(b) LiDAR time-of-flight example

Figure 2.3: Figure (a) shows the Intel RealSense camera that uses time-of-flight (LiDAR) to capture depth data. Figure (b) displays an example of time-of-flight. Images from [Cor]

way that coded light cameras would. The distance between the two sensors directly affects how far away these cameras can measure, the broader the distance between sensors, the farther the camera can capture depth.

Every type of depth camera depends on preexisting knowledge to extrapolate depth. For LiDAR, the speed of light is used to compute depth in the time of flight context. LiDAR sensors are a form of time-of-flight camera that uses laser light to determine depth. All time of flight instruments produce some light, sweep it across the scene, and then measure how long it takes for the light to return to a sensor present on the camera. The range at which time of flight sensors can measure depth depends on the light's intensity and wavelength. The main drawback of time-of-flight cameras is that they may be vulnerable to interference from other cameras in the same area and may perform less well outside. The quality of the depth image might be lower where the light striking the sensor might not have originated from the particular camera but have come from another source, such as another camera.

These cameras and sensors are used to capture and create point cloud data from the real-world. In the case of the point clouds collected using RGB-D cameras there are six values associated with each point, these values correspond to the RGB color values and the Cartesian coordinates.

Point clouds are used for many purposes: generation of object 3D CAD models, quality inspections in many different areas, visualization of a 3D space in a digital form, and augmented and virtual reality. They are especially useful in robotics to capture 3D data of a robot's surroundings and used for scene understanding.

With the widespread use of this technology, RGB-D and depth, these cameras are becoming more cost-effective and have improved in quality over the years. Low-cost consumer RGB-D Cameras provide an alternative way to solving challenges where the depth

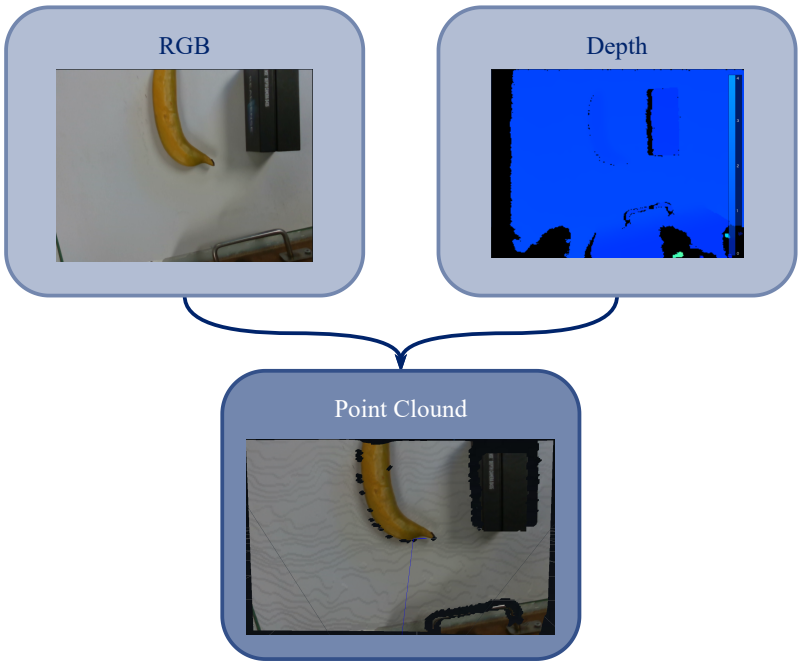


Figure 2.4: Example of RGB and Depth image and the combination of these two images into a Point Cloud.

may aid in the processing of the RGB frame. Comparing these cameras to professional laser depth scanners is unfair. They tend to be more expensive, have better quality and more field of view in comparison with RGB-D cameras like the RealSense, Kinect, and others. Even with the constraints and lower quality compared to the top-of-line hardware, these cameras are small and can aid in solving problems that require this particular data type.

2.2 Point Cloud Library

Most techniques used for 3D object recognition rely on finding characteristic keypoints (good keypoints that may represent an object) in a point cloud. Then these characteristic keypoints are matched with other characteristic keypoints that were previously stored and correctly labeled. Also, using 3D object recognition tends to be more robust to clutter than 2D (crowded scenes where objects in the top occlude objects bellow).

The Point Cloud Library (PCL) is an open, stand-alone project for 2D/3D image and point cloud processing. The PCL has a series of modular libraries. These modular libraries have multiple computer vision methods and algorithms implemented for different applications like filtering, feature extraction, segmentation, input/output (IO), visualization and many others.

In the next subsections, the focus will be on feature extraction from point clouds and present the two categories currently in the PCL project that are most relevant to the work in this document.

2.3 Descriptors

Descriptors are methods that receive data and will output relevant features that represent the original data in some way. Features are the information extracted from images and/or point clouds (more specific in the work that is presented in this document) in terms of numerical values that are difficult to understand and correlate by humans. Generally, features extracted from an image are of much lower dimension than the original image.

There are two types of features that can be extracted from the data, global and local features. These features are sometimes referred to as descriptors. Global descriptors are generally used in image retrieval, object detection and classification. Whereas the local descriptors are used for object recognition and/or identification. There is a large difference between detection and identification. Detection means finding the existence of something or an object and identification, or recognition, means finding the identity of an object.

Global features describe the image as a whole so it generalizes the entire object. These features usually include representations for contours, shapes, and textures. Local features describe the image patches of an object. For object detection and classification applications, global features are used, whereas local features are used for applications like object recognition. The combination of global and local features generally tends to improve the accuracy of the recognition with a cost of computational overheads.

For the scope of the work that will be presented, there are two relevant global descriptors that will be described:

- Ensemble of Shape Functions (ESF) [WV11];
- Viewpoint Feature Histogram (VFH) [RBTH10].

Ensemble of Shape Functions is a global descriptor that consists of 10 concatenated 64-bin histograms. It outputs a single normalized histogram with a size of 640 for a point cloud. ESF descriptor were choose from the PCL to extract the features of the point cloud generated from the depth data that is received from a RGB-D camera. The main reason to choose this global descriptor to be used is that it does not need the calculations of the surface normals and this makes it a fast global descriptor [Ale12].

Another equally relevant global descriptor is the Viewpoint Feature Histogram (VFH) which differentiates itself from other descriptors by using the viewpoint vector direction. It consists of four 45-bin histograms for the angles and distances between each point and one 128-bin histogram which represents each points' Normals. So the output of the descriptor has a single normalized histogram with 308 values for a point cloud. This descriptor was chosen because it has fewer values than ESF [Ale12] while not needing the calculations of the surface normals.

2.4 Artificial Neural Networks

Artificial Neural Networks (ANN), also referred to feedforward neural networks (FFNN) or multilayer perceptrons (MLP) [GBC16][ZLLS21], where perceptron stands for a layer of neurons, neural network unit that performs certain computations to recognise features. A diagram of the various computations performed by a neuron is represented in Figure 2.5. First, a set of inputs from x_1 to x_n is defined, which are multiplied by their respective weights (w). Then, they are added and the result of this addition is subjected to a nonlinear activation function that yields the final result ($\hat{y}$). In addition, a bias (b) is used in this process, which shifts the activation function (f) to the left or right.

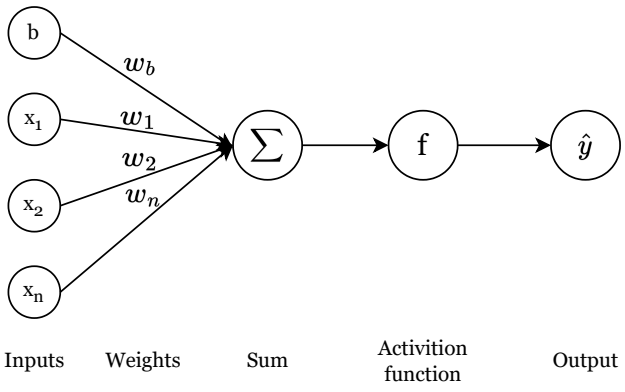


Figure 2.5: A schematic view of a neuron. The w represents the weights, b is the bias, f is the activation function, x_1 to x_n are the perceptron inputs and the $\hat{y}$ is the output of the perceptron. Image adapted from [ZLLS21]

The purpose of feedforward networks is to define a mapping between input x and output y that leads to the best function approximation by learning features θ. The models are named feedforward because the information flows directly from x to $\hat{y}$, passing through the hidden layers so that it has only one direction. They are referred to as networks because they can be seen as a weighted graph, which is usually called a network. The neurons are grouped into layers, with the first layer being the input layer, the last layer being the output layer, and the layers in between being the hidden layers. The total length of the chain thus indicates the depth of the model.

A linear model is very useful for various problems, but it also has its limitations. The basis of a linear model is that it can predict outputs as a linear function of inputs. Neural networks can use linearity, but the big advantage is that neural networks can break linearity by using nonlinear activation functions. In fact, this is the main difference between linear models and neural networks. Therefore, neural networks are trained with iterative gradient-based optimizers that bring the cost function to a low value, while linear equation solvers and convex optimization algorithms guarantee convergence regardless of the input. For example the stochastic gradient descent, which is used to find the best model parameters that correspond to the best fit between predicted and actual output, is sensitive to the values of the initial features. For feedforward neural networks, it is of great importance to initialize all weights with small random values, when training them from scratch.

A cost function (also called a loss function) is a measure of how well a model is able to make predictions for a given set of input data. The goal of training a machine learning or deep learning model is to minimize the cost function, which means finding the set of model parameters that give the lowest possible value of the cost function. There are many different types of cost functions, and the choice of which one to use depends on the specific problem being solved and the type of model being trained. For example, minimizing the mean squared error (MSE), represented in the following equation:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \quad (2.1)$$

where the MSE loss function is represented by $L(\theta)$. The model parameters being optimized are represented by θ. The number of samples in the dataset is represented by n. The ground truth target value for the *i-th* sample is represented by y_i. The predicted value for the *i-th* sample, given the model parameters θ and the input data x_i, is represented by $f_\theta(x_i)$. By minimizing the MSE loss function with respect to the model parameters θ, it is possible to find the optimal set of parameters that will give the best predictions for a given set of input data.

Another example is the mean absolute error (MAE), illustrated in the following equation:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - f_\theta(x_i)| \quad (2.2)$$

The MAE loss function is used to measure the absolute differences between the predicted and actual values. The model parameters being optimized are represented by θ, and the number of samples in the dataset is represented by n. The ground truth target value for the *i-th* sample is represented by y_i, while the predicted value for the *i-th* sample, given the model parameters θ and the input data x_i, is represented by $f_\theta(x_i)$. The goal of training a model with the MAE loss function is finding a set of model parameters that give the lowest possible value of the loss function, represented by $L(\theta)$. Similarly, by minimizing the loss function as in the MSE approach, it is possible to identify the optimal set of parameters that yield the best predictions for a given set of input data.

2.4.1 Hidden Units

There are many types of hidden units and it can be difficult to decide when to use which type. Most hidden units work as:

$$\mathbf{z} = \mathbf{b} + \mathbf{x} * \mathbf{W}^T, \quad (2.3)$$

where $\mathbf{x}$ is a vector of inputs, the weights are represented as $\mathbf{W}$, and $\mathbf{b}$ is the bias. Then the output $\mathbf{z}$ traverses an element-wise nonlinear function $g(\mathbf{z})$. The choice of activation function from this last step distinguishes most hidden units.

Sigmoid activation functions and hyperbolic tangent activation functions were the first applicable activation functions in neural networks. The sigmoid function (Equation 2.4), is a commonly used activation function in deep learning, used to introduce non-linearity in neural network models. It takes any input value x and maps it to a value between 0 and 1, which can be interpreted as a probability.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$

where, $\sigma(x)$ represents the sigmoid function and x is the input value to the sigmoid function.

Also used to introduce non-linearity in neural network models, the hyperbolic tangent function is a widely employed activation function in deep learning. It maps the input value x to a value between -1 and 1, and is symmetric around the origin. The hyperbolic tangent function is given by:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$

where, x is the input value.

Hyperbolic tangent activation function its similar to the identity function near zero, because $tanh(0) = 0$. Thus, training a deep neural network

$$\hat{y} = \mathbf{w}^T tanh(\mathbf{U}^T tanh(\mathbf{V}^T \mathbf{x})) \quad (2.6)$$

seems like training a linear model

$$\hat{y} = \mathbf{w}^T \mathbf{U}^T \mathbf{V}^T x \quad (2.7)$$

as long as the activations of the network are kept small, which facilitates the training of the model.

Another important activation function is the rectified linear unit, also known as ReLU (Figure 2.6). The ReLU activation function is:

$$ReLU(x) = max(0, x) \quad (2.8)$$

and it is easy to optimize because of the similarity to linear units. The difference between these two lies in the fact that ReLU returns the input value if it is positive, and returns 0 if the input value is negative. This results in a simple and computationally efficient non-linear transformation that can be used in deep learning models.

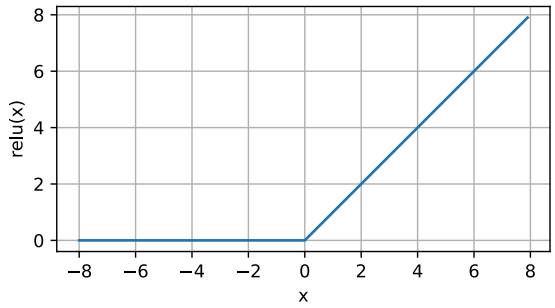


Figure 2.6: Illustration of the rectified linear unit (ReLU) [ZLLS21].

There are other hidden units that are less popular, such as radial basis function (RBF), Softplus, Hard tanh, or we can choose to have no activation function. The latter method should be used only for some hidden units, as there is a risk of creating a linear network.

2.4.2 Back-Propagation

Back-propagation is a widely used algorithm for training artificial neural networks. It involves computing the gradients of the loss function with respect to the weights of the network, and then updating the weights in a way that minimizes the loss function. The algorithm works by propagating the error backwards through the network, from the output layer to the input layer, and using the gradients to adjust the weights of the network.

The first step in backpropagation is to compute the forward pass of the network, which involves applying the input data to the network and computing the output of each neuron. The output of the network is then compared to the ground truth output, and the difference between the two is used to compute the loss function. The goal of the back-propagation algorithm is to find the values of the weights that minimize the loss function.

The second step in back-propagation is to compute the gradient of the loss function with respect to the weights of the network. This is done using the chain rule of calculus, which involves computing the gradients of the output layer and then propagating the gradients backwards through the layers of the network. The gradients are then used to update the weights of the network using an optimization algorithm such as stochastic gradient descent.

For example, consider a neural network with two layers, the gradient of the loss function with respect to a weight in the first layer can be calculated using the chain rule. The output of the first layer is used as input to the second layer, and the output of the second layer is used to compute the loss function. The chain rule states that the gradient of the loss function with respect to the weight in the first layer is the product of the gradient of the loss function with respect to the output of the second layer, the gradient of the output of the second layer with respect to the input to the second layer, and the gradient of the input to the second layer with respect to the weight in the first layer.

By applying the chain rule iteratively for each layer in the network, the gradient of the loss function with respect to all of the weights in the network can be computed. This gradient can be used to update the weights in the direction of the negative gradient of the loss function, allowing the network to learn and make accurate predictions on new inputs.

The equation for backpropagation can be represented as:

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w_{i,j}}$$

where L is the loss function, $w_{i,j}$ is the weight between neuron i and neuron j, y_k is the output of neuron k, z_j is the input to neuron j. So the back-propagation equation computes the gradient of the loss function with respect to each weight in the network, and is used to update the weights in the direction of the negative gradient of the loss function. By iteratively applying back-propagation and updating the weights, the network is trained to make accurate predictions on new inputs.

2.4.3 Architecture Design

An important point in creating a neural network model is to answer the following questions: How many units should this neural network have? How should the units be connected to each other? By answering these questions, it is possible to determine the model architecture. The units or neurons are usually grouped into layers, which are usually arranged in a chain structure, as illustrated in Figure 2.7.

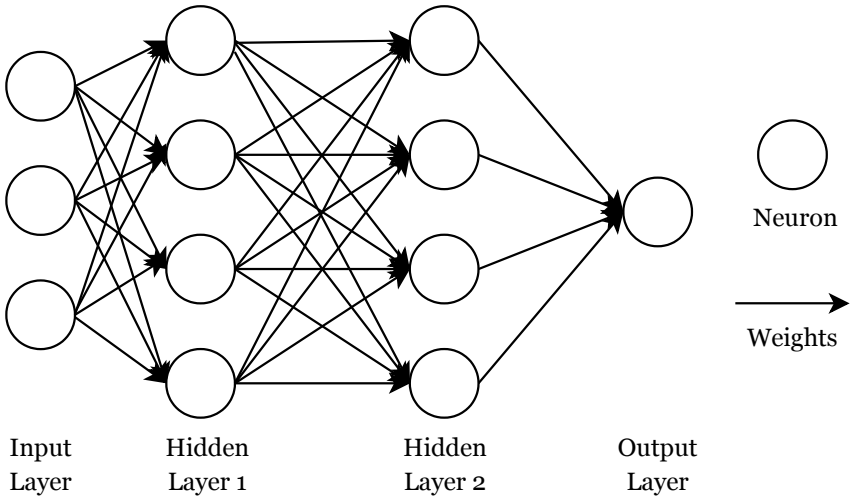


Figure 2.7: Illustration of a Multi-Layer Perceptron where it is possible to see the chain-based approach with two hidden layers. Image adapted from [GBC16]

In this architecture, each layer is a function of the preceding layer, the first layer is given by:

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{b}^{(1)} + \mathbf{W}^{(1)T} * \mathbf{x}); \quad (2.9)$$

the second layer is given by:

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{b}^{(2)} + \mathbf{W}^{(2)T} * \mathbf{h}^{(1)}); \quad (2.10)$$

and so forth. For a neural network with a chain-based structure, we must consider the width of each layer and the depth of the network. The deeper the neural network, the

more difficult it is to optimize. However, deeper neural networks are often able to use less units and thus fewer parameters with good generalization on a test set.

The goal of a neural network is to learn functions from inputs and output expected results from those functions. In a linear model, this learning is straightforward because the loss functions lead to convex optimization. MLPs, on the other hand, offered a universal system that represents all linear and non-linear functions. Nevertheless, according to the universal approximation theorem [Nis21], the training model is not guaranteed to learn the function. This happens because, first, it is possible that the optimization technique used for training cannot determine the parameter values that corresponds to the desired function. Second, the training algorithm may overfit. Using a neural network with a single hidden layer may be sufficient to represent an arbitrary continuous function. However, the layer may be impractical due to its size, and it may not learn properly leading to bad generalization on the test set. In the most extreme case, it would be necessary to use an exponential number of hidden units, with one hidden unit for each distinct input configuration. Consequently, creating deeper models may reduce generalization error by reducing the number of units required to represent a function. Thus creating the question deeper vs wider neural networks.

Using a deeper model may lead to the gradient vanishing problem. This problem derives from the activation functions used by the hidden layers and consequently its loss functions. While the model is back-propagating, it finds the derivatives of the network by moving backwards layer by layer. At the same time, each layer derivatives are multiplied in order to find the derivatives of the initial layers. The gradient vanishing problem arises when the product of this multiplication begins to decrease too much, that is, when the gradients of the loss function approach to zero. This means that the biases and weights of the initial layers will not be properly updated in each back-propagation execution, which will result in an inefficient neural network.

Adjusting the depth and width of a neural network's architecture is a common practice to improve its performance and tailor it to available computational resources. However, there is limited understanding of how these choices affect the model's accuracy.

2.4.4 Convolution Neural Networks

Convolutional neural networks (CNN) [GBC16][ZLLS21] were the algorithm that enabled a breakthrough in the world of computer vision. The term convolutional networks comes from the fact that these networks use a mathematical operation called convolution in at least one of their layers, rather than the general matrix multiplication like other neural networks. Convolution is a linear operation that uses two real valued functions to

create a third function that shows how the shape of one function is changed by the other, as shown in equation 2.11:

$$s(t) = (x * w)(t), \quad (2.11)$$

where $*$ denotes the convolution operation. The first argument (the function x) to the convolution is called input and the second argument (the function w) is the kernel, the output is known as a feature map. Equation 2.11 can be extended to ensure that time is represented as discrete:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \quad (2.12)$$

It is important to note, that a normal input of CNN are tensors and kernels are composed of a multidimensional array of parameters that are adapted by the learning algorithm. Besides that, usually convolution operations are applied to more than one axis at the time, thus the equation for two axis convolution is represented as follows:

$$s(i,j) = (w * x)(i,j) = \sum_{m}\sum_{n} x(i+m, j+n)w(m,n). \quad (2.13)$$

Equation 2.13 is a representation of function called cross-correlation, which works the same way as convolution. Neural networks in practice use this method instead of the actually convolution that requires flipping the kernel. Figure 2.8 illustrates an explanation of how basic convolution works in neural networks.

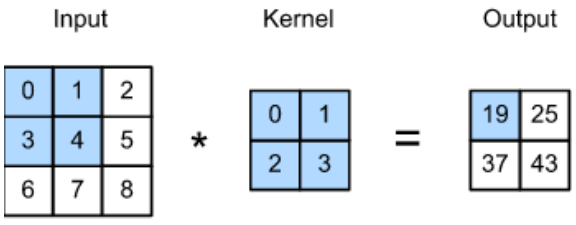


Figure 2.8: Illustration of a convolution operation, where the input tensor is a 3×3 matrix and the kernel is a 2×2 matrix. The filter or kernel will go through the matrix creating a 2×2 matrix where the entries are the results of each convolution operation [ZLLS21].

Furthermore, convolution holds three important concepts that improve neural networks learning: sparse interaction, parameter sharing and equivariant representations.

In CNNs, the interactions between input and output are far less due the size of the kernel that usually is set to be much smaller than the input, hence the name ”sparse interaction”. Moreover fewer connections means lower number of operations needed and thus less computation required. Additionally, fewer interactions also mean less parameters to be stored reducing the memory needed. A particularity caused by parameter sharing on a convolutional layer is a property called equivariance. Equivariance means if the input changes the output also changes. Translated to convolution neural networks terminology, this means that if we move the input some amount to the right then it's representation will move the same amount to the right. This is advantageous when trying to detect edges of an object in an image.

There are three stages in a CNN: convolution stage, fully connected stage and pooling stage, as exemplified in Figure 2.9. First of all, a set of linear activations are produced by the layer which operates multiple convolutions in parallel. Secondly, each linear activation is the input of a nonlinear activation function, for example, the ReLU activation function. Lastly, with the help of a pooling function the output of the layer is modified even further. The output of the network is replaced by the pooling function with a summary of the nearby outputs statistics. This means, that by using, for instance, a max pooling operation, the output will be the maximum output within a rectangular neighborhood.

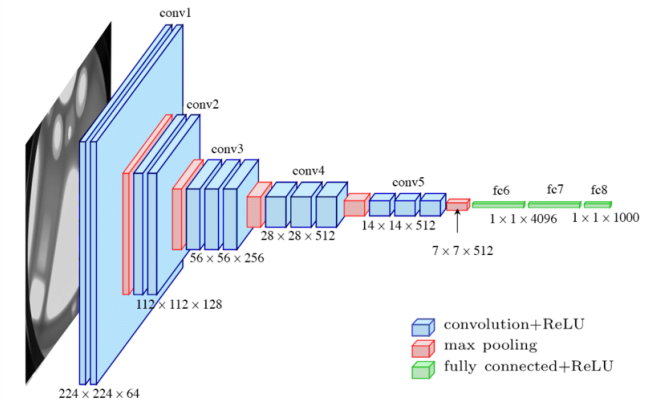


Figure 2.9: Example of the various stages on a Convolutional Neural Network architecture. The figure shows the VGG16 model architecture [FaLL17].

2.4.5 Residual Neural Network

Residual Neural Networks (ResNets) were first introduced by Kaiming He et al. [HZRS16], and have since gained popularity as a favored neural network architecture for a variety of computer vision tasks.

The primary concept underlying ResNets is the utilization of residual connections, which allow information to be directly transmitted from one layer to another. This is in contrast to conventional neural networks, where information flows only forward through the layers. By incorporating residual connections, ResNets are able to address the vanishing gradients problem. As a result, ResNets can be much deeper than traditional neural networks, leading to improved performance on an extensive range of computer vision tasks. One of the primary benefits of ResNets is their comparatively effortless training, even for very deep architectures. This is due to the residual connections which facilitate the preservation of a strong gradients throughout the network, thereby simplifying the task of updating the network's parameters during training. Figure 2.10 illustrates a residual block.

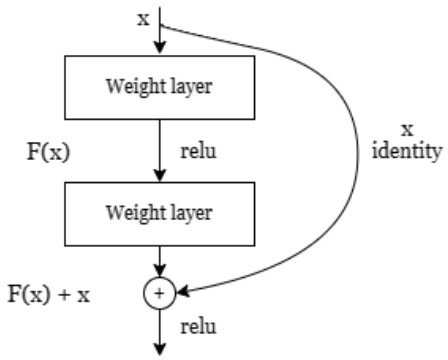


Figure 2.10: Representation of a Residual Block. Image adapted from [ZLLS21]

The output of layer l in a ResNet is given by:

$$x_l = F_l(x_{l-1}; \theta_l) + x_{l-1} \quad (2.14)$$

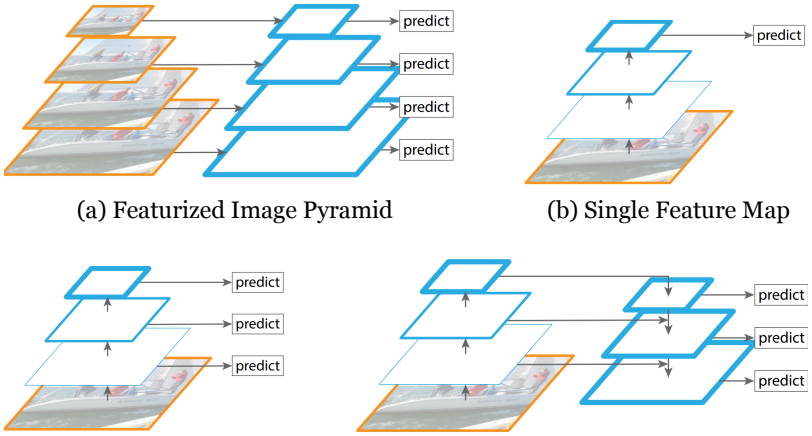
where x_{l-1} is the input to layer l. The output of the entire ResNet can be written as:

$$x_L = F_L(F_{L-1}(\cdots F_1(y_0; \theta_1) + x_0; \theta_{L-1}); \theta_L) \quad (2.15)$$

where y_0 is the input to the network and L is the number of layers in the network.

2.4.6 Pyramid Neural Networks

Pyramid neural networks [LDG+17] are a type of deep learning architecture that combines the strengths of both CNNs and MLPs. Pyramid networks are designed to handle



(a) Featurized Image Pyramid

(b) Single Feature Map

(c) Pyramidal Feature Hierarchy

(d) Feature Pyramid Network

Figure 2.11: The figure contains multiple pyramid neural network architectures. Figure (a) shows the Featurized Image Pyramid, this architecture computes the features on each of the image scales independently, but this process is slow. Figure (b) displays the Single Feature Map, this architecture was used in detection systems that chose to use only single-scale features to have faster detection. Figure (c) shows the Pyramidal Feature Hierarchy architecture, this architecture updated the Featurized Image Pyramid architecture to enable the features to be computed using convolutional neural networks. The Feature Pyramid Network is shown in Figure (d) which shows the skip connections in each level of the pyramid while keeping the different scales and multiple predictions in every level of the pyramid resulting in a fast inference as (b) and (c) and achieving better accuracy. Images from [LDG+17]

varying image resolutions and sizes, which makes them particularly useful for tasks like object detection and segmentation.

The idea behind pyramid networks is to use multiple resolutions of the input image to perform feature extraction. The network starts with the highest resolution image and performs convolutional operations on it to extract features. The output of this convolutional layer is then downsampled, and the process is repeated on the lower resolution image. This process is repeated several times, creating a pyramid of feature maps, each with a different level of resolution.

The pyramid of feature maps is then fed into a set of fully connected layers, which perform classification or regression task. The fully connected layers take the feature maps from each level of the pyramid and concatenate them together, creating a feature pyramid that captures both fine-grained and coarse-grained features.

A feature pyramid neural network (FPN) is an extension of the pyramid neural network architecture that is designed to improve performance on object detection and segmenta-

tion tasks. FPNs were proposed to address the problem of scale variation in object detection, where objects in an image can vary significantly in size.

FPNs use a pyramid of feature maps, similar to the pyramid neural network architecture. However, instead of feeding the feature maps into a set of fully connected layers, FPNs use a top-down pathway and lateral connections to combine feature maps at different levels of the pyramid. The top-down pathway consists of a series of upsampling layers that take the feature maps from higher levels of the pyramid and upsample them to the same resolution as the feature maps at lower levels of the pyramid. The upsampled feature maps are then combined with the feature maps at the same resolution using lateral connections.

The combination of the top-down pathway and lateral connections allows the network to capture multi-scale features that are important for object detection and segmentation. The high-resolution feature maps at the top of the pyramid capture fine-grained details, while the low-resolution feature maps at the bottom of the pyramid capture more coarse-grained information. The top-down pathway and lateral connections allow the network to combine these features in a way that is sensitive to scale and context.

FPNs have been shown to significantly improve performance on object detection and segmentation tasks compared to other architectures. They are particularly useful for detecting objects of varying scales in an image, which can be a challenging task for other architectures. FPNs are widely used in object detection and segmentation models, including Faster R-CNN and Mask R-CNN. Figure 2.11 shows multiple pyramid neural network architectures.

2.5 Object Detection and Localization

Object detection and localization are two related tasks in computer vision that involve identifying and localizing objects within an image or video. Object detection is the process of identifying the location and the class of objects present in an image. It is essential in applications such as autonomous driving, surveillance, and robotics, where it is necessary to detect and recognize objects in real-time. Object detection algorithms learn patterns in image data that correspond to objects of interest. These models are trained on large datasets of annotated images, where each image is labeled with the location and class of objects within it. Object localization is the process of identifying the location of a specific object within an image. It is often used in applications where it is necessary to track or manipulate a specific object. Object localization algorithms use techniques such as bounding boxes to define the spatial extent of objects in an image.

Both object detection and localization are challenging tasks due to the variability of objects in images. Objects can have different shapes, sizes, orientations, and appearances. They can also be occluded or partially visible, making it difficult to detect or localize them accurately. Moreover, the presence of background clutter or other objects in the scene can also make object detection and localization more challenging.

Deep learning models have been successful in solving object detection and localization problems by learning to recognize patterns in image data that correspond to objects of interest. These models use techniques such as CNNs and RNNs to process image data and make predictions about the location and class of objects within it.

2.6 Image Segmentation

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments, or sets of pixels where the pixels in each segment are more similar to each other than to pixels in other segments. The goal of segmentation is to simplify the images by changing their representation into sets of pixels that are more meaningful and easier to analyze. Image segmentation is typically used to classify objects, locate objects, and the object boundaries in images.

The output of image segmentation methods is a set of segments or regions that collectively cover the entire image or a set of contours extracted from the image. Each pixel in a region is similar to others within it, according to some characteristics, such as color, intensity, or texture. Adjacent regions are significantly different concerning the same characteristics.

In this document, the image segmentation methods that will be focused are based on deep learning. Deep learning image segmentation can be divided into three branches, semantic segmentation, instance segmentation, and recently, a new branch was created called panoptic segmentation.

Semantic segmentation is the process of assigning a label to, or classifying, every pixel in an image such that pixels with the same label share certain characteristics. It is possible to assign a class label to every pixel in an image because they share certain characteristics such as color, intensity, texture, and other types of characteristics.

In contrast to the semantic segmentation, the instance segmentation goal is to identify each individual object. So if in semantic segmentation all image pixels belonging to cars

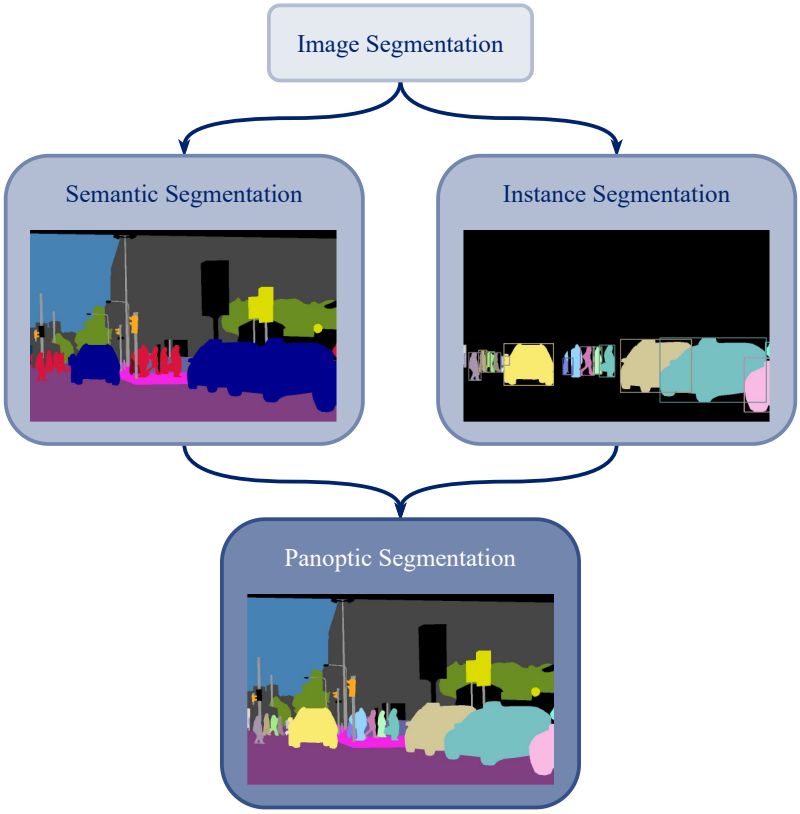


Figure 2.12: Examples of different image segmentation tasks.

were painted blue, in instance segmentation each car's pixels would be painted with a different color, effectively identifying each instance of a car.

Panoptic segmentation is the combination of the two aforementioned: semantic segmentation and instance segmentation. Kirillov *et al.* [KHG+19] introduced panoptic segmentation, where "panoptic" is defined as "including everything visible in one view" so panoptic segmentation aims to achieve a unified global view of the segmentation of the image. The objective of the panoptic segmentation task is the unification of the two typically distinct tasks of semantic segmentation and instance segmentation for a given image. So now each image pixel receives two labels: one for the type of object or a semantic label (for example, car) and another for the instance id (for example, car 1, c1a1r 2, etc.). Figure 2.12 shows examples of each type of image segmentation as mentioned above.

2.7 Data sets and Evaluation Metrics

This section introduces the data sets used throughout the thesis and the metrics used to evaluate 6D pose estimation methods as well as semantic segmentation tasks. LineMOD and YCB Video were created to evaluate methods that try to solve the computer vision 6D pose estimation task. These data sets were captured and annotated manually. Nowadays

they are the most used data sets to evaluate 6D pose estimation methods. The next two sections will have a in depth description and examples from these two data sets. In the 6D pose estimation metrics subsection of this section are described the two metrics used to evaluate the 6D pose estimation methods performance. We also introduce the most used data set for the semantic segmentation task (Cityscapes) as well the evaluation metrics used in the aforementioned task.

2.7.1 Cityscapes

The Cityscapes data set [COR+16] is used for semantic interpretation of urban street scenes. This data set is the most used data set to evaluate image segmentation methods. It has 30 classes presented in a diversity of images captured over 50 cities during all four seasons and ranging from mild to moderate weather conditions. The data set has 25000 annotated images for multiple task benchmarks like semantic segmentation, instance segmentation and panoptic segmentation. From the 25000 images present in the data set 5000 were picked to have fine annotations. The images chosen to have fine annotations have rich backgrounds and multiple objects in the foreground. The images present in the data set have resolution of 1024×2048. Figure 2.13(c) show some examples of the RGB image data present on the data set.

2.7.2 LineMOD

LineMOD [HHC+11a] is the most used data set for evaluating 6D pose estimation methods. Many types of methods that tackle the 6D pose estimation problem use this data set ranging from the classical methods like [VLM18] to the most recent deep learning approaches [WXZ+19], [XSNF17], [PA20b], [PA23]. This data set was captured with a Kinect camera, and this camera has a built-in procedure that automatically aligns the RGB and depth images. LineMOD has 15 low-textured objects (although only 13 objects are used) in over 18000 images and has the ground truth pose annotated. The 13 used objects are: ape, bench-vise, camera, can, cat, driller, duck, egg-box, glue, hole-puncher, iron, lamp, and phone. Most methods only use these 13 objects due two some missing meshes in the 3D CAD files for the other two objects and with the errors presented in these CAD files, many methods would have bad estimations due to the different shapes that these two CAD files have compared with the objects presented in the captured data. Because of this problem, most researchers choose to exclude these two objects from the data set evaluations.

For each object present in the data set there are images on the test subset that have significant clutter with other objects and mild occlusion. Figure 2.13(a) show some examples of the RGB image data present on the data set. Besides the collected data (RGB and depth

images) for each object the data set also provides the 3D model saved as point cloud data, a distance file with the maximum diameter (cm) of the objects, camera intrinsic parameters, 6D pose annotations for each known object present in each scene, and segmentation masks.

2.7.3 YCB-Video

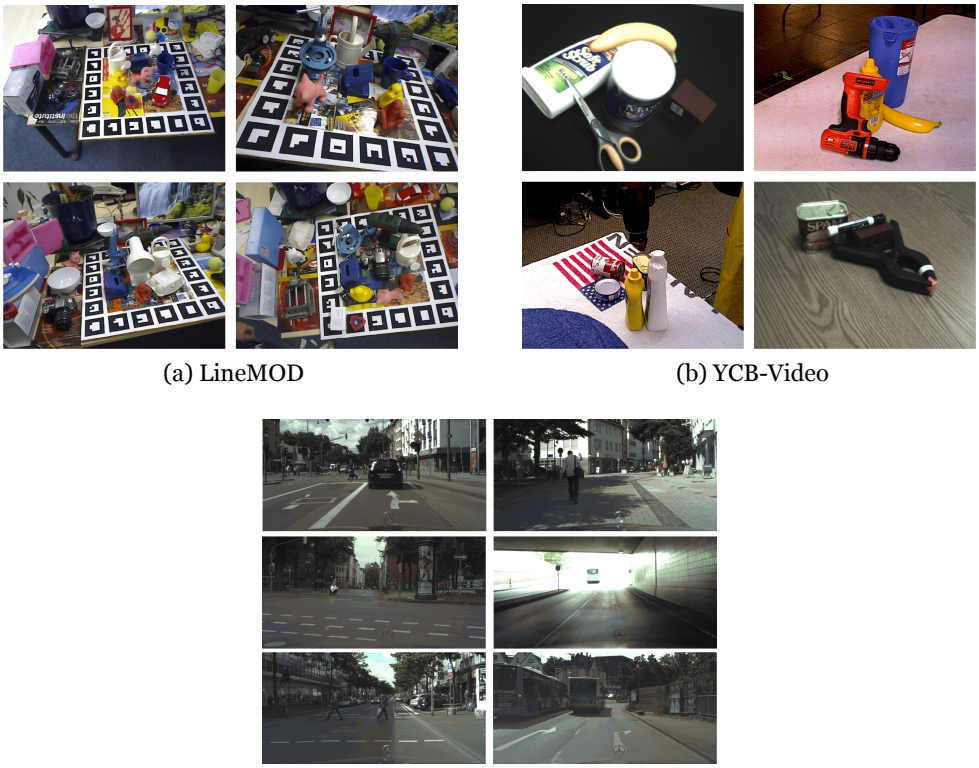
The YCB Object data set [CSW+15] provides 600 RGB and depth images, and multiple annotations for 77 objects that are divided into 5 categories: food items, kitchen items, tool items, shape items, task items. The YCB Object data set main uses are semantic segmentation, object detection and localization and object manipulation. It was initially designed for benchmarking robotic manipulation. It contains objects that appear in a daily life with different shapes, sizes, textures and rigidity, and also task items, which are objects widely used in robotic manipulation tests, objects like wooden colored cubes and other colorful wooden shaped objects.

From the YCB Object data set the YCB-Video [XSNF17] was created where only 21 objects were selected. This data set's main goal was to be used in 6D pose estimation tasks. The scenes present in the data set have heavy occlusions, and multiple known objects presented in the scene (at least three objects). YCB-Video is composed of 92 RGB-D videos with 133827 frames, each with a subset of the objects placed in the scene. These videos have the RGB and depth data, segmentation masks and the 6D poses of the objects annotated, for each frame. For each object, a 3D model file is also provided as well the camera intrinsic parameters. The 21 objects present in the data set have different shapes, textures, sizes. Figure 2.13(b) shows some examples of the RGB image data present on the YCB-Video data set.

The authors of [XSNF17] generated *80000* synthetic data with the correct annotations that were added to the data set after, but these images are only used in the train subset of the data set.

2.7.4 6D Pose Estimation Metrics

The metrics that were used to evaluate and make comparisons with other methods are the two metrics that were used in all previous works, like [KMT+17], [PLH+19], [WXZ+19], [XSNF17]. As previous authors, we use the Average Distance of Model Points (ADD) [HHC+11a] as metric of evaluation for non-symmetric objects and we use the Average Closest Point Distance (ADD-S) [XSNF17] for symmetric objects.



(a) LineMOD (b) YCB-Video

(c) Cityscapes

Figure 2.13: Example of images from the data sets.

These two metrics are needed because in the case of the symmetric objects its not possible to have specific keypoints that represent a specific point of that object, the same keypoint can be found on the other half of the object. Thus leading to the matching between points in symmetric objects (like, egg-box and glue presented in the LineMOD data set) being ambiguous for some poses.

The ADD is defined as:

$$\text{ADD} = \frac{1}{m} \sum_{x \in M} \left\| (Rx + t) - (\hat{R}x + \hat{t}) \right\|. \quad (2.16)$$

Assuming the ground truth rotation R and translation t and the estimated rotation $\tilde{R}$ and translation $\tilde{t}$, the average distance calculates the mean of the pairwise distances between the 3D model points of the ground truth pose and the estimated pose. In equation (2.16) and (2.17) M represents the set of 3D model points and m is the number of points in M.

The definition of ADD-S is:

$$\text{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \left\| (Rx_1 + t) - (\hat{R}x_2 + \hat{t}) \right\|. \quad (2.17)$$

6D Pose Estimation and Object Recognition

For the YCB-Video, besides using these metrics, we also used as in previous works like PoseCNN [XSNF17] and DenseFusion [WXZ+19], the area under the ADD-S (2.17) curve (AUC), and we also calculate the percentage of ADD-S smaller than $2cm$, which DenseFusion [WXZ+19] authors consider as the minimum error tolerance for most of the robot grippers.

Using these metrics allows us to compare directly our developed work with methods proposed by other authors.

2.7.5 Pixel-wise Accuracy

Pixel-wise accuracy is the most used metric to measure the performance of methods that tackle the semantic segmentation task. This metric consists in measuring the percentage of pixels in the image that were correctly classified. The report usually consists in the percentage per class and then an average for all classes present in the data set.

When considering the per-class pixel accuracy the ground truth consists in a binary mask for that class, where a true positive represents a pixel that is correctly predicted to belong to the ground truth mask for the given class, whereas a true negative represents a pixel that is correctly identified as not belonging to the given class:

$$Acc = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative} \tag{2.18}$$

Pixel-wise accuracy is a good metric to use when evaluating segmentation models because it provides a clear measure of how well a model is able to distinguish between different objects in an image. However, it is important to note that pixel-wise accuracy does not take into account the size or shape of the objects being segmented. The limitation of pixel-wise accuracy is that it treats all pixels in an image equally, regardless of their location or the size of the objects they belong to. Thus meaning that a model that performs well at segmenting small objects may be penalized if it has trouble with larger objects, even if it correctly segments most of the image.

For example, consider an image with a small object in one corner and a large object that occupies most of the image. A model that is able to accurately segment the large object but struggles with the smaller object may still achieve a high pixel-wise accuracy score, even though its performance on the smaller object is poor. On the other hand if the model has a good segmentation on the smaller object but fails in the segmentation of the larger object the model will achieve low score.

To address this limitation, other metrics such as Intersection over Union take into account the overlap between the predicted and ground truth segmentation masks, which can provide a more nuanced evaluation of how well a model is able to segment objects of different sizes and shapes.

2.7.6 Intersection over Union

The Intersection over Union (IoU) quantifies the percentage overlap between the ground truth mask (*target*) and our predicted output (*prediction*). It divides the total number of pixels across both masks by the number of pixels shared by the ground truth and predicted masks:

$$IoU = \frac{target \cap prediction}{target \cup prediction}. \quad (2.19)$$

The overall IoU score is calculated independently for each class and then averaged across all classes to provide a global mean IoU score for semantic segmentation.

Chapter 3

Related Work

The most significant literature on object 6D pose estimation is covered in this chapter. Firstly, an overview is given to provide a general understanding. The methods for semantic segmentation are presented in section 3.2, followed by the 6D pose estimation methods.

3.1 Overview

Recognizing the position of objects is essential for comprehending the real-world. Existing research has handled several difficulties in the 2D domain by employing approaches [GDDM14, RHGS15] that construct 2D bounding boxes of interest regions. In recent years, 6D pose estimation has received a lot of attention since 2D research cannot fulfill people's demands, which, to a large part, specifies a stiff transition from object coordinate to camera coordinate, including 3D position and 3D rotation.

Estimating the 6D pose of objects is a prerequisite for many unconstrained practical applications. Autonomous driving technologies are calculating the posture of other cars and objects to avoid collisions [CMW+17, GLU12]. Moving an object from one area to another defined position is a fundamental task of robotics [GLU12, ZDY+14], and autonomous grabbing by the robot arm may considerably increase operating efficiency. Similarly, Augmented Reality [MUS15, YWC05] can only provide users with a true experience after receiving information about their 3D position and orientation.

The objective is to determine the object's relative position and rotation. However, the recognized object's shape, size, and texture richness vary depending on the task. In an ideal circumstance, the solution would be generic for all types of objects and operate accurately and fast in various situations, including occlusion, sensor noise, and light conditions.

Existing approaches typically used two data sources: RGB images and depth data. Before introducing low-cost, high-resolution depth sensors, many methods only used RGB-D images to infer the object's pose. In a light-deprived situation, inferring a 6D position

with insufficient texturing using only RGB data was always challenging and inaccurate [PZC+17, SMD+18].

On the other hand, many methods concentrated solely on working with depth data to solve the task at hand. Many researchers converted depth data to 3D voxel grids [MS15, MAFK17] before the proposal of PointNet [QSMG17b], which eliminated superfluous processing. Literature like Frustrum PointNets [QLW+18] and VoxelNet [ZT18] used depth information successfully using the new representation. In contrast to approaches that only use one data source, the current tendency is to merge the two data sources [TTKK14, KBM+15], which provides additional information to complete 3D features. Researchers are now tackling how to harness and integrate two types of data as the data has become more varied and accurate.

Obtaining a discriminative and robust feature representation is critical for 6D pose estimation. For 6D pose estimation, there were primarily three technical solutions: traditional methods, end-to-end deep learning-based approaches, and classic methods integrated with deep learning. Traditional approaches [HCI+11, HHC+11b, Kat03] typically extract features from input data using artificially set fixed rules, followed by correspondence grouping and hypothesis verification. End-to-end deep learning methods [BKM+14, BMK+16, TTKK14] extracted features using Convolutional Neural Networks rather than handcrafting features.

These end-to-end deep learning data-driven methods for 6D pose estimation were introduced using direct regression on input images. Many researchers, however, considered that regressing directly from the input was not a reasonable practice. Therefore they chose to combine classical approaches with deep learning [TSF18, RL17], such as leveraging 2D-3D correspondences discovered by identifying 2D projection of sparse 3D keypoints.

To accurately estimate the 6D pose of an object, it is essential to separate the object from its background. One method for achieving this is semantic segmentation. Semantic segmentation allows for the identification and separation of objects within an image. By utilizing image segmentation, is possible to separate an object from its background, creating a mask that solely includes the object of interest. This mask can subsequently be employed to remove the background from the image, permitting a focus solely on the object of interest. This can lead to more accurate 6D pose estimation, as the algorithm can concentrate solely on the relevant object without being influenced by the surrounding background.

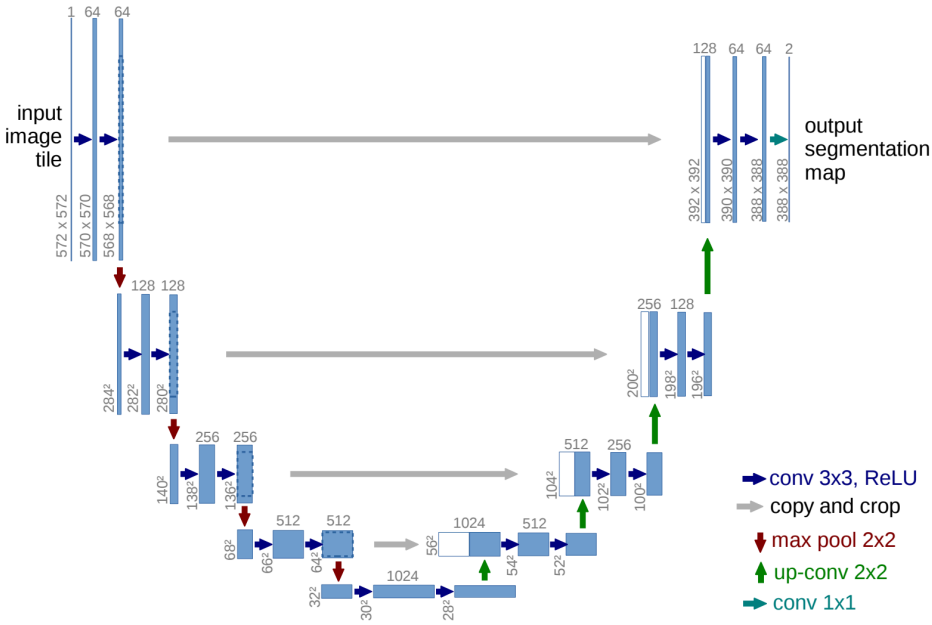


Figure 3.1: U-Net model architecture. Figure from [RFB15].

3.2 Semantic Segmentation

The semantic segmentation task aims to classify each pixel in the image. With this classification and location of each pixel, it is possible to detect and locate objects in the images, but if there are two objects of the same type cluttered together it's not possible to know that there are two objects. For this, we have instance segmentation that besides classifying each pixel of the image it detects and classifies instances of objects thus enabling the possibility to distinguish between objects of the same type in the image.

One of the most notable methods for solving the semantic segmentation task is the U-Net [RFB15] convolutional neural network that was originally developed for segmenting biomedical images. Its name comes from the architecture U letter-like shape. The U-Net architecture is composed of two parts, the left part of the U is the contracting path and the right part is the expansive path. The contracting path's purpose is to capture context while the purpose of the expansive path is to assign the classes to the pixels based on the context. SegNet [BKC17] has a similar architecture to U-Net where the second half of those architectures consists of the same structure in the first half but hierarchically opposite. SegNet, instead of using its own architecture like U-Net, it uses a fully convolutional neural network based on the VGG-16 [KSH12] convolutional layers (architecture illustrated in Figure 2.9).

FastFCN [WZH+19] architecture uses a Joint Pyramid Upsampling module instead of using dilated convolutions to assign classes to the pixels since those consume more mem-

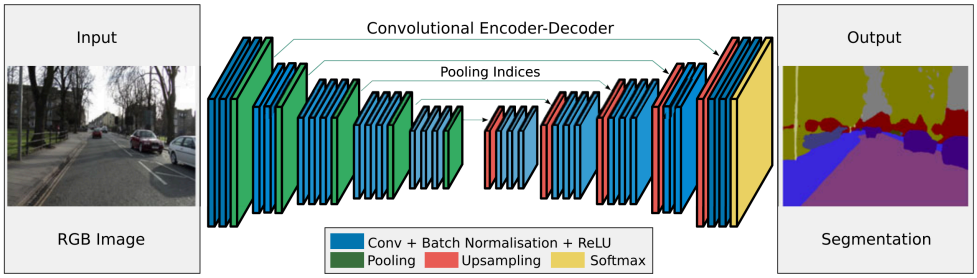


Figure 3.2: SegNet model architecture. Figure from [BKC17].

ory and time during training.

FastFCN has a fully connected network as its backbone and at the end the Joint Pyramid Upsampling is used to upsample the features and label the pixels. This pyramid upsamples the low-resolution feature maps into high-resolution feature maps. The Pyramid Scene Parsing Network (PSPNet) [ZSQ+17], uses global contextual information for semantic segmentation. The authors introduced a Pyramid Pooling Module after the last layers of a fully convolutional neural network that is based on ResNet-18, the feature maps obtained from the fully convolutional neural network are pooled using 4 different scales corresponding to 4 different pyramid levels each with different sizes, 1×1, 2×2, 3×3 and 6×6. The polled feature maps are then convoluted using a 1×1 convolution layer to reduce the feature maps dimension. These outputs of each convolution are then upsampled and concatenated with the initial feature maps that were extracted from the fully convolution neural network. This concatenation provides the local and global contextual information of the image. After the concatenation, the authors use another convolution layer to generate the final pixel-wise predictions. PSPNet objective is to observe the whole feature map in sub-regions with different locations using the pyramid pooling module. With this approach, the neural network achieves a better scene understanding.

The Gated-SCNN [TAJF19] architecture consists of a two-stream CNN architecture. The first stream branch is used to process image shape information. The second stream branch is used to process boundary information. In this architecture, a gating method is used in the intermediate layers of each branch to connect features from both branches. In the end, it uses a fusion module to fuse all the features from both branches and predict the semantic segmentation masks.

DeepLab [CPSA17] has good performances because one challenge with using deep fully convolutions neural networks on images for segmentation tasks is that input feature maps become smaller while traversing through the convolutional and pooling layers of the network. This causes loss of information about the images and results in output where predictions are of low resolution and object boundaries are fuzzy. DeepLab models address

this challenge by using Atrous convolutions and Atrous Spatial Pyramid Pooling modules. DeepLab architecture has evolved over several generations: DeepLabV1 uses Atrous Convolution and Fully Connected Conditional Random Field to control the resolution at which image features are computed. DeepLabV2 uses Atrous Spatial Pyramid Pooling to consider objects at different scales and segments with improved accuracy. Finally, DeepLabV3, apart from using Atrous Convolution, also uses an improved Atrous Spatial Pyramid Pooling module by including batch normalization and image-level features, and it does not use the Conditional Random Field as in previous versions.

3.3 6D Pose Estimation

The methods for 6D Pose Estimation can be divided into two different categories, RGB and RGB-D methods. These categories come from the type of input data that these methods receive and use in the pipelines, either only 2D data (RGB) or 3D data (RGB-D).

3.3.1 Pose Detection using RGB Images

The area of pose detection and object detection with the use of RGB images is further behind in accuracy than the RGB-D. With the use of additional information from the depth channel, it is possible to have more accuracy. However, the use of RGB images is much more common due to the low cost of the traditional cameras.

This section is divided into two subsections. These subsections are the two main different methods of estimating 6D pose with RGB images. In section 3.3.1.1 we present some methods that use deep learning for object detection and segmentation, and in section 3.3.1.2 methods that use Perspective-n-Point (PnP) or use PnP on top of the object detection deep learning methods presented in section 3.3.1.1.

3.3.1.1 Object Detection and Segmentation Methods

The methods that will be presented in this section are needed to have an accurate estimation of an object's 6D pose. Prior to the pose estimation using RGB images, it is imperative to address the challenge of segmenting or detecting the object from the cluttered environment. Therefore, as a pre-processing step, it is necessary to utilize either segmentation methods or object detection techniques to achieve the 6D pose estimation goal.

Region-based Convolutional Neural Network (R-CNN) [GDDM16] is an object detection approach that creates a number of object regions or Regions of Interest (ROI). The

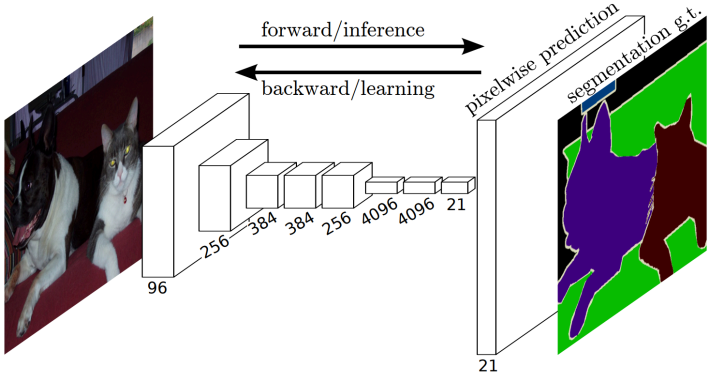


Figure 3.3: Fully convolutional neural network model architecture. Figure from [LSD15].

evolved version of R-CNN is Faster R-CNN, that uses an improved mechanism for Region Proposal Network (RPN).

RPN is a neural network used in object detection tasks in computer vision. The main goal of RPN is to generate a set of candidate regions, also known as region proposals, that potentially contain an object of interest in an input image. These region proposals are then fed into a separate network to perform object classification and bounding box regression. RPN operates on a feature map obtained from an input image by a CNN. The CNN is typically a pre-trained network, which has been trained on a large dataset. The feature map output by the CNN is typically of much lower resolution than the input image, but contains rich information about the spatial layout of objects in the image. It takes the feature map as input and applies a sliding window approach to generate a set of anchor boxes at each spatial location in the feature map. Each anchor box is associated with a set of learnable parameters that encode the offset between the anchor box and the ground truth bounding box of an object in the image. The RPN then predicts the probability of each anchor box being the location of an object of interest, as well as the corresponding bounding box offsets. The output of the RPN is a set of region proposals, each represented by an anchor box and its associated probability and bounding box offset. These region proposals are then passed through a non-maximum suppression algorithm to eliminate redundant proposals and select the top-scoring proposals.

The Faster R-CNN [RHGS15] performs object detection in two phases. First, it determines the object's bounding box, and then the ROIs are determined, with an RPN. In the second phase, each ROI has its given object class label. This is done with a method called ROI pooling.

The Mask R-CNN [HGDG17] is a simple, flexible and general structure for object instance segmentation. It can detect objects in an image and at the same time generate a

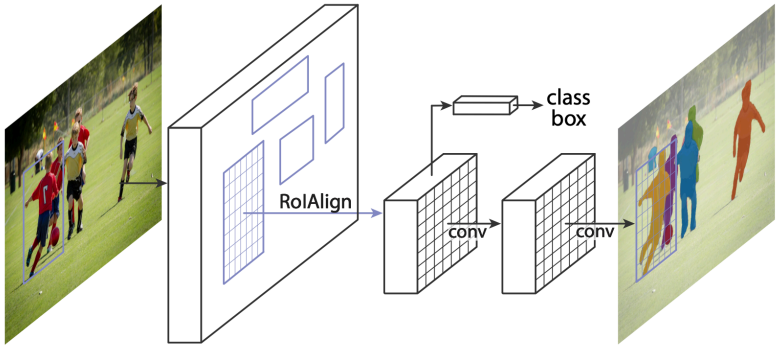


Figure 3.4: Mask R-CNN model architecture. Figure from [HGDG17].

targeting mask for each instance. It has two stages to solve two sub-problems, one for Object Detection that uses an architecture similar to Faster R-CNN [RHGS15], and another for the problem of semantic segmentation which uses a CNN [LSD15].

Mask R-CNN [HGDG17] incorporates the methods from the Faster R-CNN, but there is a data loss problem in ROI pooling. The ROI pooling problem, usually involves the max pooling in a ROI bounding box, thus quantizing the slip. The quantised stride method works by considering a ROI of 17×17 and then it needs to be mapped to a space of 7×7, hence the required stride is $17/7$ which is 2.42. Since a stride of 2.42 is meaningless, the ROI polling will quantise this value by rounding it down to 2, so it will use a stride of 2 along the width and the height. However, in doing so, it only considers the top 14×14 pixels in the 17×17 region. The remaining points are lost. Not only is there a loss of data but this can also lead to misalignment. If we use a 18×18 input and map it to a 7×7 output, the required stride becomes 2.57 and this rounds up to 3 in ROI Pooling so there is also a misalignment when we perform polling.

To address the ROI Pooling problem, ROI Align is used and non-quantisation takes place so, in the case of the 17×17 input region, we consider a 2.42 stride as it is. However this value is meaningless since each cell is divided into a 2×2 bin so that creates 4 regions in the top left, the top right, the bottom left and the bottom right, and each of these sub cells is pulled through linear interpolation leading to 4 values per cell and the final cell value is then computed by either an average or the maximum over the 4 sub values. By addressing the loss and misalignment problems of ROI Pooling, ROI Aligned has better results than ROI Pooling and it preserves spatial pixel to pixel alignment for every ROI and there is no information loss, as there is no quantisation.

Conceptually, the R-CNN mask is similar to the faster R-CNN, but the R-CNN mask also displays the object mask using pixel-to-pixel alignment. This mask is a binary mask issued for each ROI. In Semantic Segmentation, a convolutional network is used to predict the

mask of each ROI. Convolutional layers are used because they maintain spatial orientation and this information is crucial for specific localisation tasks, such as creating an object mask.

Pre-detection systems redirect classifiers or locators to perform the detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered to be detections. Yolo [RF18] uses a totally different approach. Applying a single neural network to the complete image. This network divides the image into regions and provides bounding boxes and probabilities for each region. The bounding boxes are weighted by the predicted probabilities.

The YOLO architecture has several advantages over classifier-based systems. It analyses the entire image so that its predictions are informed by the overall context in the image. It also makes predictions with a single network assessment, unlike systems like R-CNN, which require multiple ROIs for a single image. This makes the YOLO architecture extremely fast.

3.3.1.2 Perspective-n-Point Methods

Brachmann [BMK+16] *et al.* created a three-phase pipeline structure for object pose detection. They begin on the first phase with a random forest that predicts object labels and object coordinates for every pixel of the input image. To reduce the uncertainty of the predictions as much s possible the authors extended the random forest to an auto-context random forest (ACRF).

An ACRF is a machine learning algorithm that combines the strengths of both random forests and auto-context methods. Random forests are an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. Auto-context methods, on the other hand, use previous iterations of a model's output as input to the next iteration to improve accuracy. In an ACRF, multiple random forests are trained using different subsets of the training data. Each tree in a given forest is trained to predict a specific output label, based on a set of input features. The output of each tree is then fed back into the input data as an additional feature, and the process is repeated multiple times. The goal of this method is to improve the accuracy of the final output by incorporating context information from previous iterations. By doing so, the ACRF is able to capture complex relationships between the input features and output labels, resulting in improved performance compared to traditional random forests.

6D Pose Estimation and Object Recognition

On the second phase the poses of multiple objects are predicted from the 2D and 3D correspondences using the preemptive RANSAC. Preemptive RANSAC is a variant of the RANSAC, which is an iterative method for estimating the parameters of a mathematical model from a set of data points that may contain outliers. In the standard RANSAC algorithm, a subset of data points is randomly selected to estimate the parameters of the model, while the remaining data points are considered outliers. The algorithm then determines how many of the remaining data points fit the estimated model within a certain threshold, and this process is repeated until a satisfactory model is obtained. In contrast, the preemptive RANSAC seeks to reduce the amount of computation required by the standard RANSAC algorithm by using a preemptive mechanism to stop the algorithm early if it detects that the current estimate of the model is unlikely to improve significantly with additional iterations.

On the third stage, the poses are refined by exploiting the uncertainty of the object coordinate predictions.

Oberweger [ORL18] *et al.* proposed a method that can estimate the pose of an object, even if the image has occlusions. They start with the prediction algorithm that moves in a sliding window to search for regions of interest (ROI). When a ROI is recieved, a patch is created (a Patch is a segment of an image with a ROI centred on it). In these patches, they predict the heatmaps with a deep neural network. A heatmap is a graphical representation of data in which specific values in the image are represented as different colours. In this case, the corners of the object are marked as the hottest points in the images. This helps in delineating the bounding box of the object. After the prediction, they take all the heatmaps and extract the global maximum for each of them, and then the pose of the 3D object is calculated with the *PnP* algorithm.

Rad [RL17] *et al.* has developed a method for detecting the object pose from RGB images. Segmentation is used to detect objects and, after segmentation, a CNN is used to create bounding boxes and estimate the pose of the object. They begin by identifying the 2D centre of the objects of interest in the input images. They created a method based on image segmentation that performs well because it can provide accurate locations even under occlusions. After obtaining the centred image, they predict the 3D pose of an object by applying a deep neural network in the image window centered on the 2D object. In segmentation, they used an architecture similar to VGG [SZ14] that served as the basis for this method. Image segmentation provides the identity and 2D locations of the visible objects, and then the 3D pose can be estimated from the matches between the 3D points of the object and the bounding box provided by the *PnP* algorithm.

3.3.2 Pose Detection using RGB-D Images

In this section, we present some techniques that are used in pose detection when using RGB-D images.

Nowadays the use of RGB-D cameras is more accessible than ever, given their relatively low cost. One of the main requirements in this thesis is to detect objects and obtain their 6D pose to know the best possible way to perform grasping. To achieve this objective, an accurate object detection and object 6D pose is needed. This is useful to later allow a robot to learn how to execute pick-and-place tasks of known objects in an unconstrained environment.

This section has four subsections, each one has an overview of the state-of-the-art separated in each category methods.

3.3.2.1 Pose Estimation using 3D Global Information

When the background of a given scene is a known, it's easy to segment foreground objects from the background. Imagine that several objects are placed on a table. If a dominant plane is found on the table by fitting a 3D plane model, separating the objects from the table is a straightforward task. This segmentation scheme significantly reduces the search space for the objects, and this type of planar background assumption has been widely adopted in robotic manipulation. In [AVB+11], the authors devised the clustered viewpoint feature histogram (CVFH) which is an extension of the viewpoint feature histogram (VFH) [RBTH10] to 6-DOF pose estimation on segmented object clouds. The authors of [LBRF11] proposed a tree structure for scalable object recognition and pose estimation, but the pose estimation is limited in that it can only estimate the coarse 1-DOF rotation of the object pose. Although these approaches can recognise an object pose efficiently, they depend on perfect segmentation from the background. These approaches are only applicable to the well-structured table-top manipulation, and are not robust in cluttered scenes.

Reliable depth information is preferred to find correspondences in visually featureless objects, so it is used in registration and tracking problems. The iterative-closest point (ICP) [BM92] algorithm is well-known for the registration of 3D point clouds, but it requires a good initial pose estimate to converge to the global optimum. To enhance the ICP algorithm, beam-based models [HHRF11] were proposed. These type of models were further improved by exploiting over-segmentation. For environment mapping, [NIH+11] presented a real-time mapping and pose tracking system, which fuses a live depth stream into a volumetric scene model called the truncated signed distance function (TSDF) and

tracks the camera pose with respect to the scene model using the ICP algorithm. For object tracking, several works using point clouds have been proposed using particle filters. A particle filter-based tracking in the Point Cloud Library (PCL) [RC11] can track the 6-DOF pose of a reference point cloud model over a sequence of RGB-D images by the using point and colour likelihood functions.

Zeng [ZYS+17] *et al.* presented an approach that leverages multi-view RGB-D data and self-supervised data-driven learning. They segment and label multiple views of a scene with a Fully Convolutional Network (FCN) [LSD15], and then fit pre-scanned 3D object models to the resulting segmentation to get the 6D object pose. Their FCN has a VGG architecture [SZ14] to perform the object segmentation and the weights are initialised using a pre-trained model on ImageNet for 1000-way object classification. They fine-tune the network over the 40 class output classifier (39 Classes for each Amazon Picking Challenge (APC) object and 1 class for background) using stochastic gradient descent with momentum. They also proposed a self-supervised method to generate a large labelled data set without manual segmentation. Their semi-automatic data gathering system is done by taking pictures of the empty bins from many different viewpoints to get the background of each bin. After having the background of each bin, the objects are put inside the bins in random poses and the robot camera takes pictures from the same viewpoints as in the background collection. To obtain pixel-wise object segmentation labels, they create an object mask that separates the foreground from background. The system takes in an RGB-D image from multiple views of the scene and outputs 6D poses. It also outputs a segmented point cloud. The robot uses the segmented point cloud to complete its tasks (picking and stowing). The robot camera is positioned on the gripper and points to the object. After getting the segmented point cloud, their algorithm works in two phases: first, it segments the RGB-D point clouds captured from multiple views into different objects using a CNN. Second, it aligns pre-scanned 3D models of identified objects to the segmented point clouds to estimate the 6D pose of each object. Iterative Closest Point (ICP) [GIRL03] is the algorithm used on the segmented point cloud to fit pre-scanned 3D models of objects and their pose is estimated. Their algorithm performs poorly on products that are meshed, transparent or have plastic wrapping. The depth in this type of objects is too noisy or does not have any good values. Their solution can still work even on the hard cases stated above because they use a multi-view segmentation to estimate the convex hull of the object.

PoseCNN [XSNF17] created a CNN for 6D object pose estimation that achieved end-to-end 6D object pose estimation and it is robust to occlusions. The authors also introduced ShapeMatch-Loss, a new loss function to improve their CNN on objects that are symmetric. PoseCNN performs three tasks in an end-to-end configuration. First, they start by predicting an object label for each pixel in the input image. Second, they estimate the 2D pixel coordinates of the object centre by predicting a unit vector from each pixel towards the centre. Using semantic labels the image pixels associated with an ob-

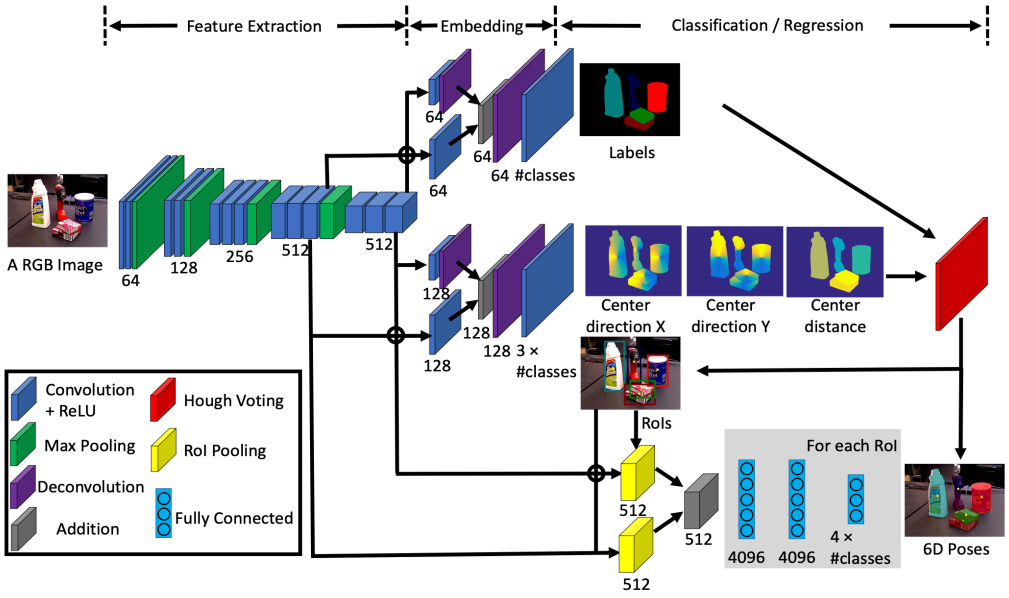


Figure 3.5: PoseCNN model architecture. Figure from [XSNF17].

ject vote on the object centre location in the image. With the knowledge of the camera intrinsic parameters, estimation of the 2D object centre and its distance make possible to recover its 3D translation. On the final step, the 3D rotation is estimated by regressing convolutional features extracted from the inside of the bounding box of the object to a quaternion representation of the 3D rotation. This method can be used on textured objects or texture-less objects, and it can detect partially occluded objects if the network is trained to vote on object centres, however, their algorithm cannot handle multiple object instances of the same category. One of the hardest challenges of the pose estimation is symmetric objects, objects which in different orientations can generate identical observations. So they created a new loss function called ShapeMatch-Loss, which is focused on matching the 3D shape of an object and it has superior performance in objects that are symmetric. Their network architecture for 6D object pose estimation contains two stages. The first stage consists of 13 convolutional layers and 4 max-pooling layers, which extract feature maps with different resolutions from the input image. It is the backbone of the network since the extracted features are shared across all the tasks performed by the network. The second stage consists of an embedding step that embeds the high-dimensional feature maps generated by the first stage into low-dimensional, task-specific features. In the end, the network performs three different tasks that lead to the 6D pose estimation: semantic labelling, 3D translation estimation, and 3D rotation regression. They use semantic labelling to detect each object in the image, as the network classifies each pixel into an object class. This method provides a better output than the algorithms that use object detection with bounding boxes and it can also handle object occlusions. In the end, they can use depth images to do pose refinement using the ICP algorithm.

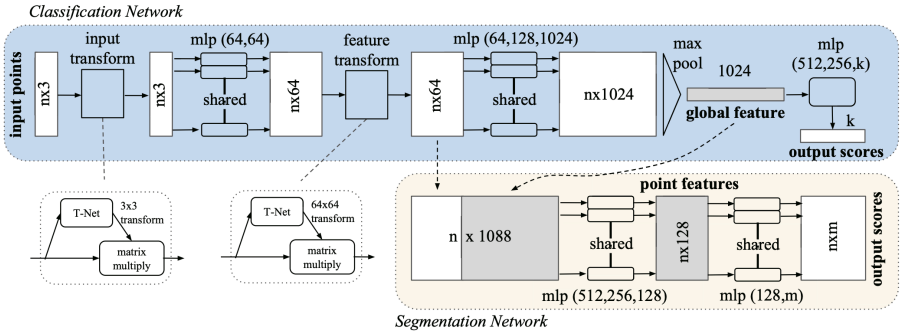


Figure 3.6: PointNet model architecture. Figure from [QSMG17a].

3.3.2.2 Pose Estimation using 3D Local Information

When no prior knowledge about the background structure exists, a set of object models has to be matched directly with a given scene. Similar to 2D local image key-point descriptors, several 3D point descriptors have been proposed based on the distribution of the surface normals around a point, surface curvature, spin image, relative angles between neighbouring normal's [RBTH10], and neighbouring range values around an interesting point [SRKB11]. While these features are invariant to a rigid body transformation, they are in general sensitive to noise, meshed objects and resolution difference of point clouds. Moreover, stable point descriptions on geometrically simple objects are limited, and hence estimating pose only with these point descriptors might not be a reliable solution.

Kehl [KMT+16] *et al.* presented a 3D object detection method that uses regressed descriptors of locally-sampled RGB-D patches for 6D vote casting. They employed a convolutional autoencoder (CAE) that was trained on a large collection of random local patches. In the testing phase, scene patch descriptors were matched against a database of synthetic model view patches and cast 6D object votes which are subsequently filtered to refine the hypotheses. It uses a holistic approach and delivers impressive results in terms of object retrieval and pose estimation, although it can not be directly applied to object detection in cluttered scenes since a precise object localisation is needed.

The authors of [KMT+16] trained a CAE from scratch using random patches from RGB-D images with the goal of descriptor regression. The network created codebooks from synthetic patches sampled from object views where each codebook entry holds a local 6D pose vote. In the detection phase, sample patches were used as input image on a regular grid to compute their descriptors and match them against the codebooks with an approximate k-NN search to be more efficient. For the CAE they used multiple layers of 5×5 convolutions and PReLUs before a single fully-connected encoding/decoding layer. They used one max-pool operation after the first convolution to introduce a small shift-invariance and used a deconvolution with learned 2×2 kernels for up-scaling before using again

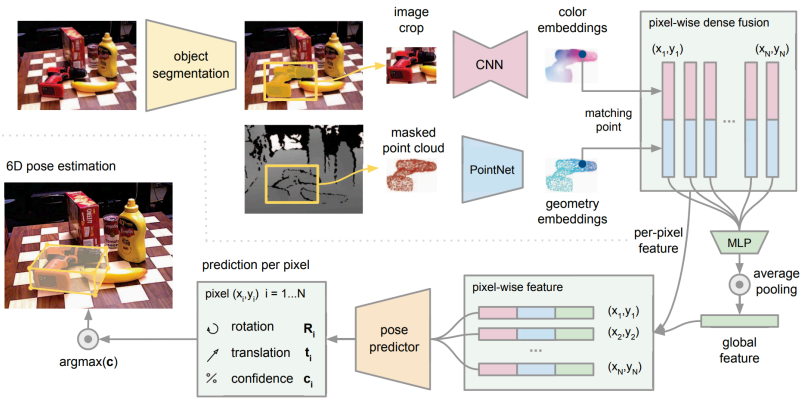


Figure 3.7: DenseFusion model architecture. Figure from [WXZ+19].

5×5 convolutions and PReLUs. The CAE learns a mapping from the high-dimensional patch space to a much lower feature space.

The authors of [SYJ18] propose a two-stage paradigm that starts from a CNN object detection in an unstructured environment and uses generative sampling to perform 6D pose estimation. Their algorithm has to estimate the pose for each object given an RGB-D image under a cluttered environment. Their goal for the stage one method is to provide object bounding boxes with confidence scores given an object class. To achieve this, they use a CNN inspired by region proposal networks (RPN) in [RHGS15]. However, instead of only classifying objects as object or non-object, the network is able to produce the object class labels. The purpose of the second stage is to estimate the object pose and further refine the bounding box, based on the estimated pose. They achieve this by performing a generative sampling-based local search. Local search is inspired by sampling methods, such as bootstrap filter, which offer robustness and versatility. The result of the first stage may be imperfect, so they expect that in the second stage the localisation of the object is improved, or even correct some false detection's, based on the result of the first stage.

DenseFusion [WXZ+19] is an end-to-end deep network model for 6D pose estimation from RGB-D images, which performs fast and accurate predictions for real-time applications. Their innovation is the fusion of the RGB values and point cloud in a per-pixel level. This is new because previous works usually use image crops to compute global features or bounding boxes. The fusion scheme enables their model to explicitly reason about the local appearance and geometry information, which is essential to handle heavy occlusion. An iterative method was made which performs pose refinement within the end-to-end learning framework. The model generates object segmentation masks and bounding boxes from RGB images and the RGB data, and point clouds from the depth map are encoded and fused at each corresponding pixel. The pose predictor produces a pose estimate for each pixel and the predictions are voted to generate the final 6D pose prediction of the

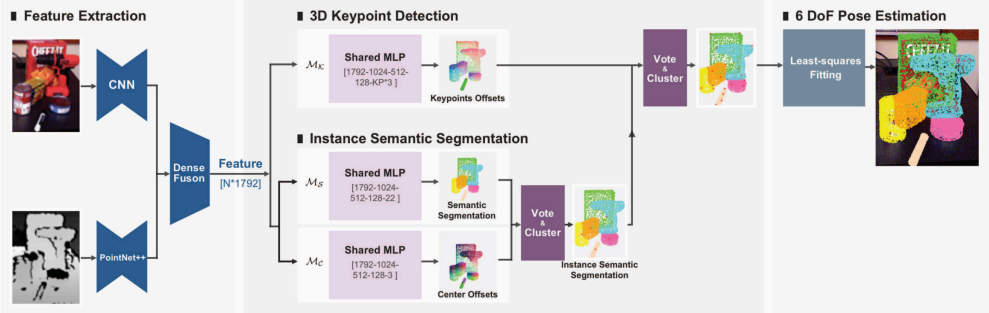


Figure 3.8: PVN3D model architecture. Figure from [HSH+20].

object. This method has two stages one that we will call pre-processing and the second that is the core stage. On the pre-processing stage, the authors perform a semantic segmentation and for each segmented object, they find in the image they will feed, one mask of depth pixels and an image patch cropped by the bounding box of the mask. The core stage has four NNs. When these NNs are chained they output a refined pose prediction. The first NN is a fully convolutional network that processes the colour information and maps each pixel in the image patch to a colour feature embedding. The second is a PointNet-based [QSMG17a] NN that processes each point in the masked 3D point cloud to a geometric feature embedding. The third NN, is a pixel-wise fusion network that combines embeddings, and outputs the estimation of the 6D pose of the object based on an unsupervised confidence score. Lastly, an iterative self-refinement NN is also presented. Iterative Closest Point (ICP) is a powerful refinement algorithm and it is widely used, however, it is not fast enough. So a new learning-based RGB-D iterative refinement method was also proposed in [WXZ+19]. They used a NN architecture that enhances model performance while keeping the inference speed in real-time. Their NN gets as input the previously mentioned predicted pose as an estimate of canonical frame of the target object and transforms the input point cloud into an estimated canonical frame.

PVN3D [HSH+20], uses the DenseFusion as a backbone to extract features of the object and then uses a shared MLP to estimate the object's keypoints and segment each object. A clustering algorithm is then used to find the different points of the object. In the end, the authors use a least-squares fitting algorithm to estimate the 6D pose of the object.

3.3.2.3 Pose Estimation using Pair Features

A pair of primitive features is a more general way to describe characteristics of the shape of objects since a set of pair features can encode both local and global shape descriptions. In addition, the pair features are inclusive in that they can represent geometrically salient areas. [DUNI10] adopted a pairing feature using two points and their normal's. In the learning phase, a set of pair features from an object is calculated and saved in a hash table

for fast retrieval. In the testing phase, points are randomly sampled from the sensor data, and each pair matched with pairs in the learned model, casts a vote for a pose hypothesis. After the voting process, a set of votes over a certain confidence level is aggregated to form possible object pose hypotheses. This approach was enhanced by considering the object boundary information [CTT+12]. The point pair feature was also used in a RANSAC-based sampling approach in [PHP+12].

3.3.2.4 Pose Estimation using Templates

There are efforts to solve the pose estimation problem via augmented template matching approaches in which depth information is used for better recognition performance. Park [PGBP10] *et al.* exploited the depth information to detect edges from depth discontinuities and adopted a template matching of the signed distance transformed images. Navab [HCI+11] *et al.* combined the image gradient from RGB channels and the surface normal vectors calculated from the depth channel. Their work employs a large number of templates for each object to take into account the variability of shape, and it uses the modern instructions for parallel computation in CPU. While their initial work learns object templates, the later work of [HLI+12] automatically generates templates from 3D mesh models. Although it can rapidly detect objects from RGB-D scenes, it is not free from the limitations of template matching, with high false positive rates, low pose accuracy, and unreliable detection outside the learned depth range.

The authors of [KBM+15] developed a method for 6D pose estimation that only uses a single RGB-D image. They use analysis-by-synthesis as a core method. They call it learn to compare because the idea is to compare the observation with the output of a forward process. A forward process is a rendered image of an object in a particular pose. To achieve the results they use a Convolutional Neural Network (CNN) inside a probabilistic context. Given the extra depth channel in RGB-D, it becomes feasible to extract the full 6D pose of object instances present in the scene, but this method, like many others, has trouble dealing with occlusions and sensor noise because the surface of the object will be different from the rendered one. Their CNN was one of the first to be used as a probabilistic model to learn and compare rendered images and observed images. They also observed that their CNN does not specialise in the geometry or appearance of specific objects, it can be used with objects of vastly different shapes and appearances and different backgrounds.

3.4 Conclusion

In this chapter, we reported the most significant related work for the work that will be presented in the following chapters of this document. In table 3.1, we present results ob-

Table 3.1: Quantitative evaluation of methods present in the related work of 6D pose estimation. Tested on the LineMOD data set and evaluated with the ADD 2.16 metric.

Type of Methods	RGB		RGB-D				
Methods	BB8 [RL17]	PoseCNN +DeepIM [XSNF17, LWJ+18]	Implicit +ICP [SMD+18]	SSD-6D +ICP [KMT+17]	PointFusion [XAJ18]	DenseFusion [WXZ+19]	PVN3D [HSH+20]
Average Accuracy	62.7	88.6	64.7	79	73.7	94.3	95.4

tained from some of the related work methods that we tested in the LineMOD [HHC+11a] data set and using the ADD metric 2.16.

RGB-D methods incorporate depth information along with color information to estimate 6D pose. This additional depth information is beneficial in every scenario, specially when using deep learning methods, it adds more information to extract features. However, RGB-D methods require specialized hardware such as depth sensors or structured light cameras, which increases the cost and complexity of the system. On the other hand, RGB methods rely solely on color information to estimate 6D pose. This makes them more accessible as they only require standard RGB cameras that are commonly available. However, RGB methods are overall less accurate when compared with RGB-D methods (Table 3.1).

Chapter 4

6D Pose Estimation Proposals

This chapter presents proposals for 6D pose estimation pipelines. The objective of the proposal pipelines is to use raw data from a RGB-D camera and obtain the known objects 6D pose as output. To achieve our objective multiple deep learning methods are required in order to locate, segment and classify the objects to further in the pipeline estimate its pose.

Foreword on Author Contributions

The results of this work have been disseminated in the form of five papers.

- [PA19] N. Pereira and L. A. Alexandre, "Using pcl gobal descriptors in a densefusion architecture," in 25th Portuguese Conference on Pattern Recognition (RECPAD), 2019.
- [PA20b] N. Pereira and L. A. Alexandre, "MaskedFusion: Mask-based 6d object pose estimation," in 19th IEEE International Conference on Machine Learning and Applications (ICMLA 2020), December 2020.
- [PA20a] N. Pereira and L. A. Alexandre, "Exploring the impact of color space in 6d object pose estimation," in 26th Portuguese Conference on Pattern Recognition (RECPAD), 2020.
- [PA21] N. Pereira and L. A. Alexandre, "Impact of segmentation and color spaces in 6d pose estimation," in IEEE International Conference on Autonomous Robot Systems and Competitions, (Santa Maria da Feira, Portugal), 2021.
- [PA23] N. Pereira and L. A. Alexandre, "MPF6D: Masked Pyramid Fusion 6D Pose Estimation," in Pattern Analysis and Applications, 2023.

4.1 Using PCL Global Descriptors in a DenseFusion Architecture

In this section, a new architecture is proposed that is based on the DenseFusion [WXZ+19] method. This new architecture replaced the feature extraction phase of the DenseFusion

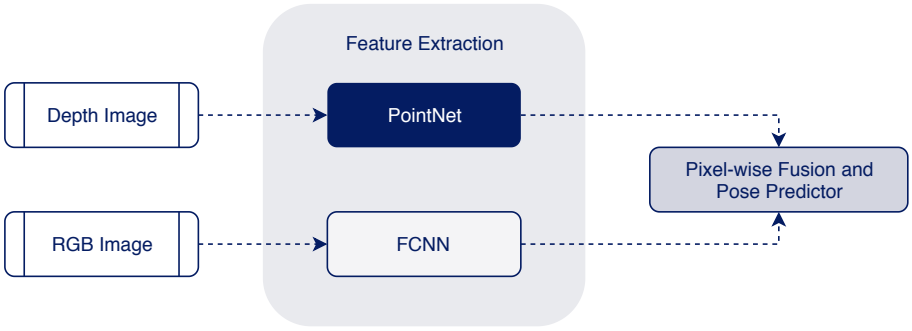


Figure 4.1: DenseFusion feature extraction phase.

with PointCloud Library (PCL) global descriptors. Instead of using the PointNet neural network architecture, the global descriptors from the PCL were used to extract features. Two global descriptors used were: Ensemble of Shape Functions (ESF) and Viewpoint Feature Histogram (VHF). Comparison results between all of the previously mentioned feature extraction approaches are presented below.

4.1.1 DenseFusion Architecture

At the time of this work DenseFusion [WXZ+19] architecture was one of the best to estimate the 6D pose of an object. This neural network architecture receives an RGB image patch that contains an object in it and also receives a point cloud data that represents the same patch as the RGB image. This architecture starts with feature extraction from both types of data. For the RGB image, a fully convolutional ResNet18 [HZRS16] is used. While for the point cloud data, a neutral network named PointNet [QSMG17a] was used to extract the features. PointNet is a deep learning neural network which provides a unified architecture for applications ranging from object classification and part segmentation, to scene semantic parsing. In Figure 4.1, the feature extraction from both types of data is shown. After the feature extraction, this method performs a fusing step of 3D data with 2D appearance features while retaining the geometric structure of the input space. The authors present results where this method outperforms PoseCNN [XSNF17] on the YCB-Video data set without the post-processing. DenseFusion, in its base, is more similar to PointFusion [XAJ18]. In PointFusion, geometric and appearance information is fused in a heterogeneous architecture. When all the features are fused, a FCNN is used to regress the object's pose.

In Figure 4.2, a diagram is shown where a complete pipeline is presented. This pipeline can be divided into five phases where three of those phases belong to the DenseFusion architecture. To use the DenseFusion architecture, image object patches are needed to estimate its 6D pose. This pipeline creates a full environment that can be used in the real-world to get raw data from an RGB-D camera to the 6D pose estimation. The first phase

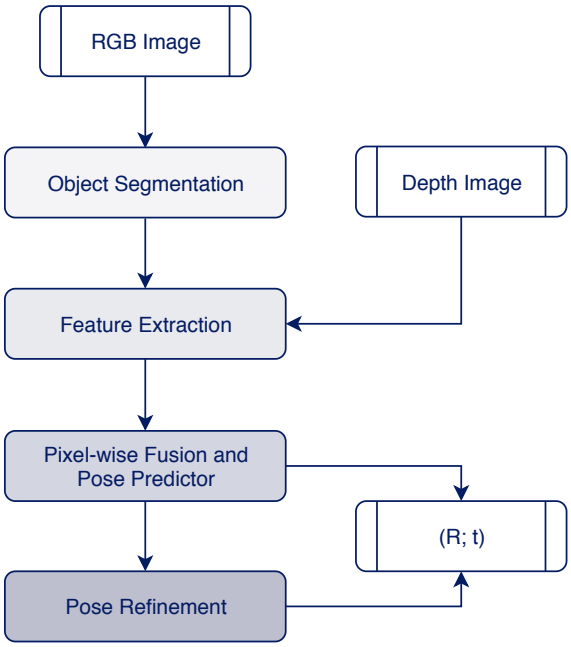


Figure 4.2: Overview of a pipeline that uses DenseFusion architecture to receive raw data from a RGB-D camera and estimate the object's 6D pose.

is where the method receives the raw RGB data and applies object segmentation or object detection to get the patches that have the object contained in it. During the second phase, the feature extraction (this phase is the first step of the DenseFusion architecture), is where features are extracted from the RGB images and point cloud data. After the feature extraction, in the third phase, both the features of the RGB images and the point cloud are fused (second step of the DenseFusion architecture). In the forth phase, and immediately after the pose predictor (third step of DenseFusion architecture) estimates the pose of an object presented in the scene, giving as output the rotation matrix and the translation vector. Finally, on the last phase, the pose refinement may be executed. It predicts small errors and adjusts the object's 6D pose to return the final pose estimation. It should be noted that this last phase, the refinement phase, is optional. The pose refinement neural network is an alternative method of Iterative Closest Point (ICP). Both methods calculate the error between a point in a 3D space and its ground truth. Furthermore, these methods try to estimate errors and generate a new 6D pose, while taking into account the errors.

4.1.2 Global Descriptors in a DenseFusion Architecture

After studying the DenseFusion architecture, a few improvements were idealized. One of them is presented in this section. The new approach that was proposed, is a modification in the architecture of the DenseFusion. In the feature extraction phase, instead of using the PointNet method, it uses PCL global descriptors to extract features from the point cloud of the objects and then fuses those features with the RGB features. The main

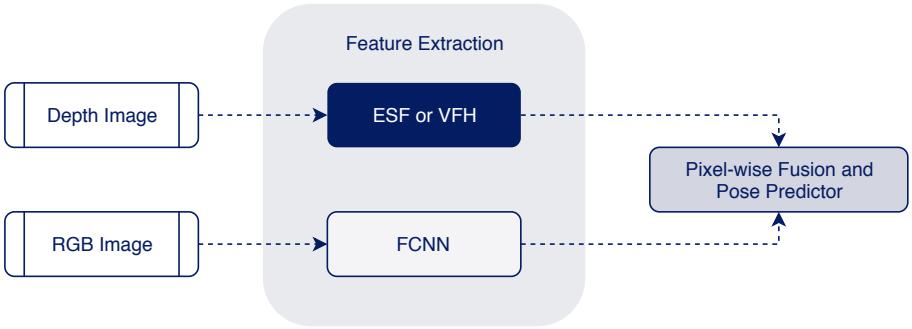


Figure 4.3: Modifications on the feature extraction phase. Replacement of PointNet to the ESF or VFH descriptor.

reason to choose global descriptors is the fixed size of the features vector. Two global descriptors (Ensemble of Shape Functions [WV11] (ESF) and Viewpoint Feature Histogram [RBTH10] (VFH)) that were available in PCL were chosen to replace the neural network which does the feature extraction from the point cloud data. The descriptors in Figure 4.3, ESF and VFH, replaced the PointNet feature extractor from Figure 4.1. Using descriptors required changes in the fusion step of the DenseFusion. The fusion layers needed to be adjusted to receive different size of features from the output of each descriptor, and then fuse them with the RGB extracted features.

4.1.3 Experiments

For this work the LineMOD [HHC+11a] data set was used to compare the original DenseFusion architecture to the proposed architectures that were modified to use global descriptors to extract features from the point cloud data. All experiments were executed on a desktop with SSD NVME, 64GB of RAM, an NVIDIA GeForce GTX 1080 Ti and Intel Core i7-7700K CPU.

The experiments relied on training three different methods for the point cloud feature extraction (PointNet, ESF and VFH). Each experiment was trained for 50 epochs with tests occurring in epoch 10, 20, 30, 40 and 50. These tests were done on the test subset of the LineMOD data set. This subset is only used for inference to estimate the object pose and measure the error from the predicted pose to the ground truth. The calculated errors from the test subset are not used and do not affect the training of the neural network. All the experiments were executed without the pose refinement phase presented by the DenseFusion authors. Since the scope of these experiments was to study the impact of using descriptors instead of neural networks in the point cloud feature extraction, the optional pose refinement phase was not taken into account. Using the pose refinement phase would improve the accuracy of the resultant poses but in a proportional form as the obtained results. Thus, avoiding spending extra time to train the pose refinement neural network.

Table 4.1: Average error and standard deviations in millimeters and average time to train in hours.

Method \ Epoch	Average Error (Standard Deviation) [mm]					
	10	20	30	40	50	Avg. time to train (Stand. Deviation) [h]
DenseFusion using PointNet [QSMG17a]	12.9 (0.5)	12.6 (0.5)	12.4 (0.4)	12.5 (0.4)	12.8 (0.3)	42 (0.4)
DenseFusion using ESF [WV11]	13.4 (0.4)	13.0 (0.2)	12.9 (0.1)	13.1 (0.4)	13.2 (0.4)	34 (1.1)
DenseFusion using VFH [RBTH10]	13.1 (0.3)	13.0 (0.3)	12.8 (0.4)	12.7 (0.3)	12.9 (0.3)	34 (1.1)

All the hyperparameters were kept as they were in the original DenseFusion architecture, batch size one, 0.0001 as learning rate value and the Adam optimizer.

The experiment results obtained are presented in Table 4.1. An average of five different train and test runs were done to obtain the presented results. The average and standard deviation were calculated to obtain the behaviour of the methods. Since the neural networks start with random weights, it is possible to have different results from each run. The original DenseFusion architecture uses the PointNet as feature extractor from point clouds and it has the best error around $12mm$. Although it is also the feature extraction method that increases the training time of the pipeline the most. When global descriptors like ESF and VFH are used, they process the data and output the unique features of the objects so the pipeline needs less time to be trained. ESF and VFH methods had pose error around $13mm$, but using them took 8 hours less to train all the pipeline compared to using the original DenseFusion architecture. Thus, representing a 19% faster approach. In terms of inference time, the described pipeline took $14ms$ to estimate the 6D pose of an object presented in the image while using ESF or VFH compared to $15ms$ for PointNet.

Figure 4.4 presents the error values in millimeters along with the number of training epochs. Analysing Figure 4.4, it is possible to conclude that after 30 epochs, the original DenseFusion architecture (using PointNet) starts over-fitting. This was discovered because after 30 epochs the average error starts increasing for the test subset. During the experience where VFH was used to extract features from the point cloud of the object, the neural network only entered in over-fit after epoch 40. Whereas in the ESF experience, it started over-fitting at epoch 30. This was the same epoch as the original DenseFusion architecture and overall it had worse performance compared to VFH.

4.1.4 Ablation Studies

Following the same idea described above, other experiments that did not achieved the expected results will be described in this section. There were four extra experiments. The

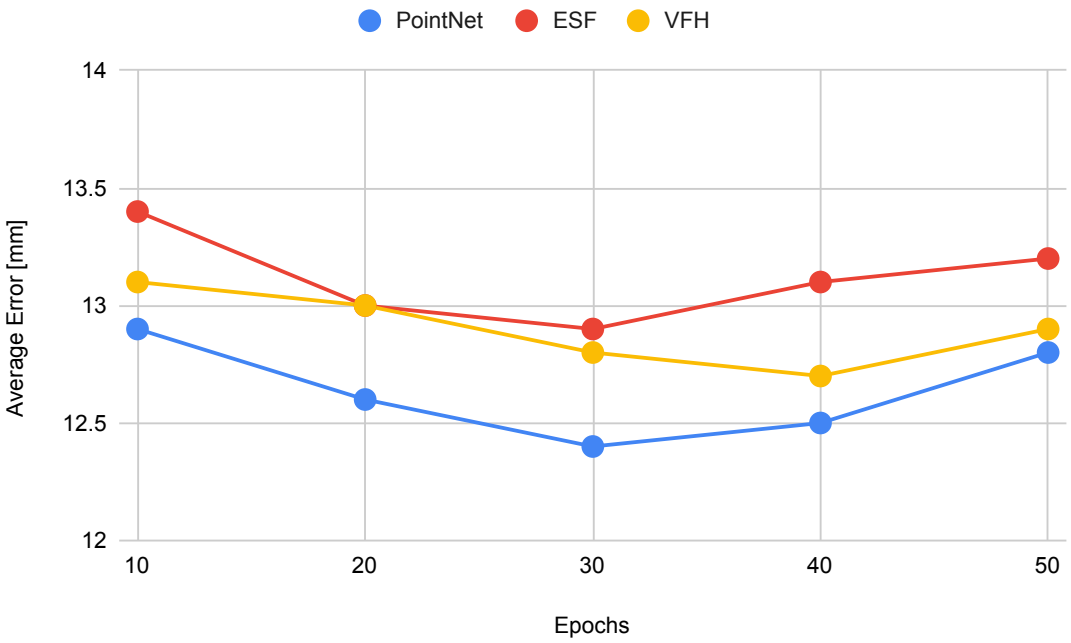


Figure 4.4: Average error in millimeters tested in different epochs.

first one, was to test if a fully convolutional neural network could extract meaningful information from the depth image instead of generating a point cloud from the depth image and extracting features from the generated point cloud. For this first experiment, a modified ResNet18 was used where the first layer of that architecture was replaced, to receive images with just one channel, and the classification multilayer perceptron (MLP) was removed, since only the convolutional layers were required to extract the features that will be used by the DenseFusion pose estimation. The other three experiments were based on concatenating features from two methods. Combining features from ESF and VFH, ESF and modified ResNet18, and finally, VFH and modified ResNet18.

Table 4.2: Point cloud feature extraction experiments. Reference Average Errors and Errors are shown in millimeters.

Method \ Epoch	10	20	30	40	50
Reference Average Errors [mm]					
DenseFusion using PointNet [QSMG17a]	12.9	12.6	12.4	12.5	12.8
DenseFusion using ESF [WV11]	13.4	13.0	12.9	13.1	13.2
DenseFusion using VFH [RBTH10]	13.1	13.0	12.8	12.7	12.9
Errors [mm]					
DenseFusion using ResNet18 [HZRS16]	25.3	24.2	24.1	25.8	30.6
DenseFusion using ESF + VFH	14.3	14.0	13.5	13.8	14.1
DenseFusion using ESF + ResNet18	37.2	34.9	34.7	34.7	36.9
DenseFusion using VFH + ResNet18	36.7	36.3	35.1	34.7	35.6

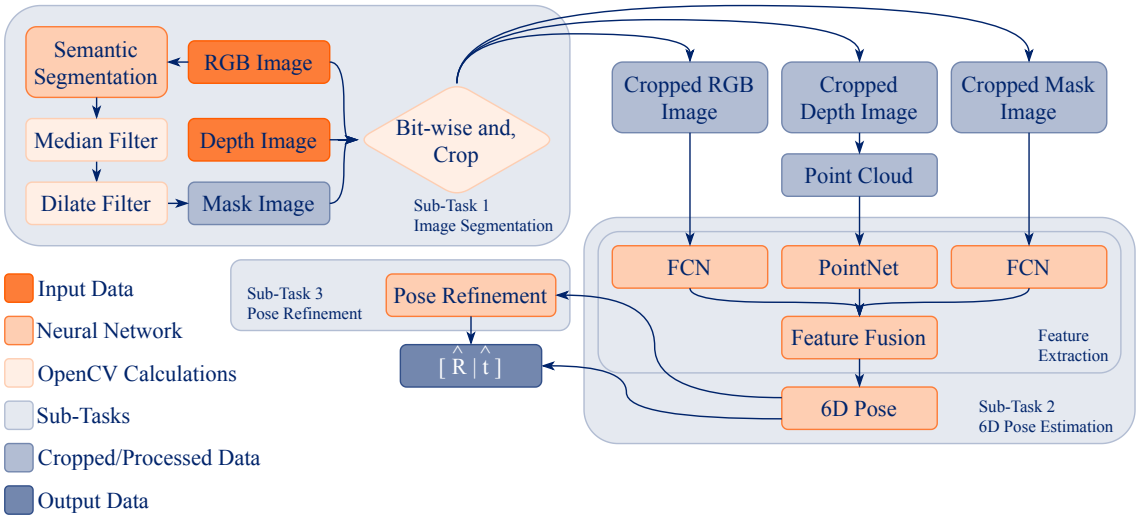


Figure 4.5: Pipeline diagram of the MaskedFusion architecture. MaskedFusion has three sub-tasks: image segmentation, 6D pose estimation, and a pose refinement.

In Table 4.2, the results for these four experiments are shown. The Reference Average Errors are the average errors reported on Table 4.1. These values are the baseline for comparison of the next four experiments. In the other table section, Errors, the errors for the four experiments are presented. Analyzing all these errors it is possible to conclude that all of these four experiments have worse performance. From these four experiments, the best method was the feature combination from the ESF and VFH, but even this method required more computation time, it had worse performance than only using one of those methods.

4.2 MaskedFusion: Mask-based 6D Object Pose Estimation

In this section, MaskedFusion, a mask-based 6D object pose estimation method is presented. MaskedFusion is a method which estimates the 6D pose of objects using RGB-D data, with an architecture that leverages multiple sub-tasks in a pipeline to achieve accurate 6D poses. Achieving accurate 6D poses aids in other open problems such as robot grasping or positioning objects in augmented reality. MaskedFusion is an improved method over the previous because it uses object masks to eliminate non-relevant data. With the inclusion of these masks on the neural network that estimates the 6D pose of an object, it is possible to have features that represent the object shape taking into account when estimating the pose of the object. MaskedFusion is a modular pipeline where each sub-task can have different methods that achieve the objective.

4.2.1 MaskedFusion Architecture

MaskedFusion comprises three distinct sub-tasks that, when integrated, can estimate an object's 6D pose. For each sub-task, a neural network is used to solve that particular task. However, since the pipeline is modular, every sub-task can have different types of methods that will solve the task at hand and they can be easily replaced.

In order to better understand the description of each sub-task of the MaskedFusion, a diagram that represents the method's flow is introduced in Figure 4.5.

The method starts by receiving RGB-D data as input. The first sub-task segments the input scene to detect and localize a known object, and generate a mask for the known object. To achieve this, the neural network for the semantic segmentation task is based on an encoder-decoder architecture. In this sub-task, only RGB data is used for inference. During training, the ground truth mask is needed to compare with the predicted masks. The use of semantic labeling provides richer information about the objects and handles occlusions better than object detection and localization methods. To train this neural network, the cross entropy loss (4.1) was used:

$$L_{CE} = -\sum_{i=1}^{n} gt_i \, log(p_i) \quad (4.1)$$

where the number of classes is represented as n, gt_i is the ground truth label, and p_i is the softmax probability for the i^{th} class.

The cross entropy loss is a commonly used loss function for the image segmentation task, because each predicted class probability is compared to the ground truth class and a score is calculated that penalizes the probability based on how far it is from the ground truth value. The penalty is logarithmic in nature, yielding a large score for large differences close to 1 and a small score for small differences tending to 0. The aim is to minimize the loss, that is, the smaller the loss the better the model should be. The perfect model has a cross-entropy loss of 0.

The use of semantic segmentation was due to its performance in terms of speed and results to solve the task at hands. Since the pipeline is modular in this sub-task, it is possible to use instance segmentation or panoptic segmentation methods to achieve the same outcome. However there will be an increase of the time needed for the pipeline to execute. This time increase is due to high computation costs that these algorithms have compared to semantic segmentation.

6D Pose Estimation and Object Recognition

After having the predicted mask for each object presented in the scene, two filters are applied to the output masks. The first filter used is a median filter to smooth the mask with a kernel size of 3×3. Second, a dilate filter is applied to the mask with a 5×5 kernel such that if the mask has some minor boundary segmentation error, this operation helps to correct it. Both filters used are available on the *Python OpenCV* library.

After the masks were generated and had the filters applied from the first sub-task, the second sub-task is almost ready to be initiated. For the second sub-task, the RGB, depth and mask data is necessary. The main goal for the second sub-task is to extract features of the data and estimate the 6D pose of the object in the scene. However, the second sub-task only uses patches or crops of isolated objects and to achieve this, a pre-processing needs to be done. In this pre-processing step, the binary mask obtained from the first sub-task method with the 2 filters applied is then used to crop the RGB and depth images. This is done for each object present in the scene. To crop the RGB and depth images, a *bit-wise and* operation is executed between the RGB image and mask, and also between the depth image and the mask. The result of the *bit-wise and* of the RGB image and the mask image, will be put inside of a rectangular crop that encloses all of the object and this smaller image will serve as input data to the next sub-task. In the case of the depth image, a point cloud is further generated from the resultant *bit-wise and* cropped depth image, and the point cloud will also serve as input to the next sub-task of the pipeline. Cropping the data with the mask is a pre-processing of the data that helps the 6D pose estimation neural network on the second sub-task, by discarding the background or other non-relevant data that are around the object leaving only the data that is most relevant to the 6D pose estimation, the objects that are being analyzed.

Once the RGB image, the mask image, and the point cloud data of each object are isolated, then the second sub-task can start by extracting features from each input data type. The 6D pose neural network sub-task can be divided into two stages, like the convolutional neural networks. The feature extraction stage and 6D pose estimation stage, as in common CNNs has a feature extraction stage where convolution layers are used and then the regression MLP predicts the outputs.

For the feature extraction, different methods were used to address the different data types. Starting with the RGB, a fully convolutional ResNet18 was used to extract features from the selected object in the scene. The use of fully convolutional ResNet18 comes from trimming the original ResNet18, the MLP was removed from it just to keep the convolution layers responsible to extract features. This same concept is also used on the mask image. For the mask image, a similar fully convolutional ResNet18 was used, whereas the first convolutional layer receives only one channel instead of three. The point cloud is were another neural network architecture was needed to extract its relevant features. The

neural network architecture used was based on the PointNet [QSMG17a]. It extracts 500 features that represent the depth information of the object.

Compared with previous methods, such as DenseFusion [WXZ+19] and PointFusion [XAJ18], having features from the mask enabled us to have features that represent the shape of the object. This is relevant information for the next stage where all information is fused and enabled us to improve the 6D pose estimation accuracy.

On the second stage, all the extracted features of each data source are then concatenated into a single tensor and pass through convolutional layers to fuse all the features. Then after all the features are fused into a single tensor. The feature tensor is then feed-forward to three neural network heads. Each neural network head has 4 convolutional layers where each one is responsible for a specific output. One head outputs the predicted translation vector that will have the Cartesian coordinates of the center point of the analysed object. The second head will output the rotation matrix that has the information of the object rotation in consideration of the viewpoint of the object. the third head outputs a confidence value for the estimation. This value can be used as in object detection and localization methods where a threshold may be applied to filter low confidence estimations. To summarize, after the second sub-task, MaskedFusion already outputs an object 6D pose estimation.

To train the 6D pose estimation neural network the following loss function (4.2) is used to calculate the error between M randomly sampled points and the ground truth object pose:

$$\mathcal{L}_i^p = \frac{1}{M} \sum_j \left\| (Rx_j + t) - (\hat{R}_i x_j + \hat{t}_i) \right\| \quad (4.2)$$

where, x_j denotes the j^{th} point of the M randomly selected 3D points from the objects 3D model, $p = [R|t]$ is the ground truth pose, where R is the rotation matrix of the object and t is the translation. The predicted pose generated from the fused embedding of the i^{th} dense-pixel is represented by $\hat{p}_i = [\hat{R}_i|\hat{t}_i]$ where $\hat{R}$ denotes the predicted rotation and $\hat{t}$ the predicted translation.

After obtaining the predicted translation vector and rotation matrix, it is possible to use another method to further refine the estimated 6D pose. This is the last sub-task and it is optional, but advisable for better pose estimation results.

6D Pose Estimation and Object Recognition

In the last sub-task, MaskedFusion uses the refinement neural network that DenseFusion [WXZ+19] presented. This last step can be slow during the training because it needs many training epochs to show its advantages. However, during the inference, it is a very fast method. Other methods, such as PoseCNN [XSNF17], usually use iterative closest point (ICP), which is also a good method for refinement, but it has a higher computational cost and will take longer to apply the refinements during inference time.

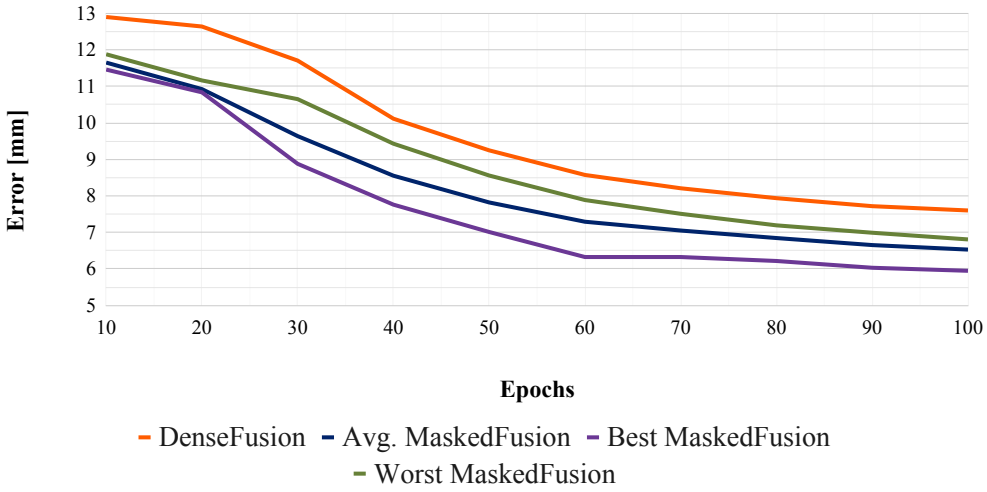
4.2.2 Experiments

MaskedFusion was tested on two widely-used data sets, LineMOD [HHC+11a] and YCB-Video [XSNF17]. These data sets were chosen, because they are widely used by the other methods and by using the same splits of the data set enable a direct comparison between MaskedFusion and previous methods. In each experiment presented, MaskedFusion was trained from scratch with the weights initialized as random, and then the evaluation was done on the test subset of the described data set. The same training procedure was done five times and the average results were presented, whereas other references usually only present the best results and without describing how many epochs of training they executed. All experiments were executed on a desktop with SSD NVME, 64GB of RAM, an NVIDIA GeForce GTX 1080 Ti and Intel Core i7-7700K CPU.

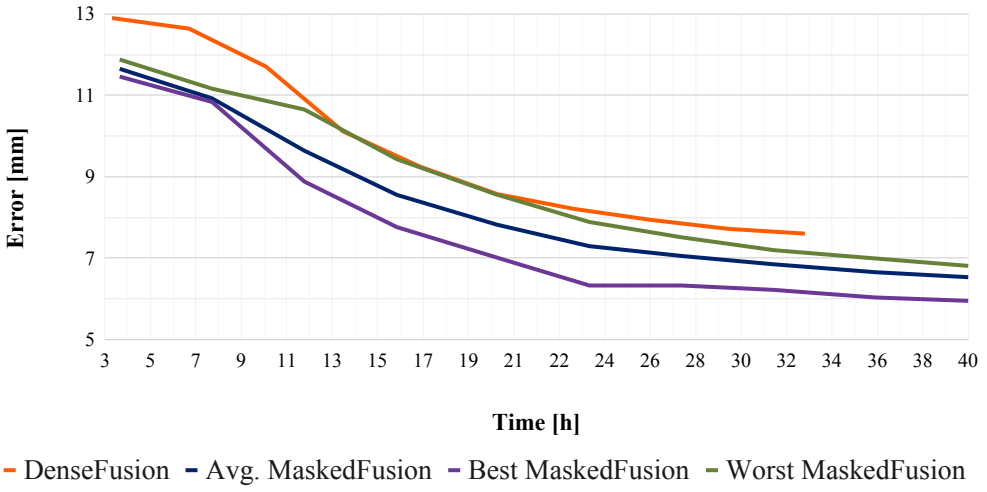
4.2.2.1 LineMOD Experiments

The MaskedFusion experiments and its results for the LineMOD data set are described in this subsection. There will be many direct comparisons with DenseFusion, because it was the best method at that time.

Figure 4.6 (a), shows the test results of DenseFusion compared with the MaskedFusion average, best and worst runs for several different epochs using the LineMOD data set. Both MaskedFusion and DenseFusion were trained for 100 epochs, and in every 10 epochs they were executed in inference mode to test them on the test subset of the LineMOD data set, and their overall object average errors were plotted. The values obtained with the inference mode were not used during training of the neural network. Analysing the plot presented in Figure 4.6 (a), it is possible to see that even the worst results from MaskedFusion are better than the best results of DenseFusion. All MaskedFusion average error values were always below the DenseFusion in all epochs tested, and most important is that MaskedFusion entered first in the $10mm$ error mark. It achieves the $10mm$ error, on average, in epoch 30 and DenseFusion only achieved this error in epoch 40. Comparing the average error of the best run to the DenseFusion best result, it had an error of $5.9mm$ and DenseFusion has $7.6mm$. MaskedFusion achieved more overall accuracy leading to possible better placement of objects in a virtual scene or better grasping accuracy.



(a) Error per epoch.



(b) Error per hour.

Figure 4.6: The methods were evaluated on the test set, after every 10 training epochs, and the figure contains the average error in millimeters. Figure (a) shows the error as a function of the training epoch whereas figure (b) presents it as a function of training time. All MaskedFusion runs had smaller error than DenseFusion.

Table 4.3: Quantitative evaluation of 6D pose using the ADD metric on the LineMOD data set. Symmetric objects are presented in italic and were evaluated using ADD-S. Bold shows best results in a given row.

				MaskedFusion		MaskedFusion				
Objects	SSD-6D + ICP	PointFusion	DenseFusion	Avg	Stdev	Individual Experiments				
ape	65.0	70.4	92.3	92.2	4.1	**97.1**	86.7	90.5	91.4	95.2
bench vi.	80.0	80.7	93.2	98.4	1.1	98.1	**100.0**	97.1	99.0	98.1
camera	78.0	60.8	94.4	98.0	1.0	97.1	97.1	**99.0**	**99.0**	98.0
can	86.0	61.1	93.1	97.4	2.3	99.0	98.0	95.0	**100.0**	95.0
cat	70.0	79.1	96.5	97.8	1.5	96.0	98.0	97.0	**100.0**	98.0
driller	73.0	47.3	87.0	95.6	1.7	95.0	96.0	93.0	**97.0**	**97.0**
duck	66.0	63.0	92.3	94.0	3.0	**97.2**	89.6	95.3	92.5	95.3
eggbox	**100.0**	99.9	99.8	99.6	0.5	**100.0**	**100.0**	99.1	99.1	**100.0**
glue	**100.0**	99.3	**100.0**	**100.0**	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
hole p.	49.0	71.8	92.1	97.3	2.5	93.3	97.1	98.1	**100.0**	98.1
iron	78.0	83.2	97.0	97.1	1.3	95.9	**99.0**	97.9	96.9	95.9
lamp	73.0	62.3	95.3	99.0	1.0	98.1	**100.0**	**100.0**	98.1	99.0
phone	79.0	78.8	92.8	98.8	1.3	97.1	**100.0**	**100.0**	99.0	98.1
Average	76.7	73.7	94.3	97.3	0.3	97.2	97.0	97.1	**97.8**	97.5

To train 100 epochs in MaskedFusion, it takes 40 hours, compared with 33 hours for DenseFusion. These times were measured using the same settings and computer. Since MaskedFusion has one more neural network to train, it has additional computation over the data, it is normal to take longer in the overall training. Note that since it can achieve a smaller error before DenseFusion in terms of training epochs, this increased training time can be disregarded. Even if the training was stopped after a fixed number of hours instead of after a fixed number of epochs, MaskedFusion would still produce a smaller estimation error, as can be demonstrated in Figure 4.6 (b). For instance, MaskedFusion entered in the $10mm$ error mark after 12 hours of training, while DenseFusion needed 13.2 hours.

In Table 4.3, a comparison of the MaskedFusion test results in a per-object comparison with three other methods: SSD-6D [KMT+17], PointFusion [XAJ18] and DenseFusion [WXZ+19]. The values presented in Table 4.3 result from the ADD metric (2.16) and ADD-S metric (2.17). From Table 4.3, it is possible to conclude that MaskedFusion has overall better accuracy than all previous methods in the LineMOD data set.

The average results from the 5 repetitions of MaskedFusion were better than DenseFusion in 11 out of 13 objects. In the worst-performing experience, the second column of MaskedFusion Individual Experiments in Table 4.3, achieved an average of 97%, which is better than the DenseFusion. Finally, the best of 5 repetitions improves the overall ADD from the 94.3% of DenseFusion to 97.8%. At the time, for the LineMOD data set, MaskedFusion achieved overall better results than all previous methods.

Table 4.4: Quantitative evaluation of 6D pose (area under the ADD-S (2.17) curve (AUC)) on the YCB-Video data set. Bold numbers are the best in a row and underline numbers are the best when comparing MaskedFusion with DenseFusion both with 100 training epochs. The last column of the table is the evaluation of MaskedFusion using the masks that were generated by our first sub-task during the train and test. (*) The values presented were obtained from [WXZ+19]

	PointFusion	PoseCNN +ICP	DenseFusion	MaskedFusion		MaskedFusion		DenseFusion		Pipeline
Training Epochs	–	–	–	200		100		100		100
Objects	AUC(*)	AUC(*)	AUC(*)	AUC (Avg)	AUC (Stdev)	AUC (Avg)	AUC (Stdev)	AUC (Avg)	AUC (Stdev)	AUC
002_master_chef_can	90.9	95.8	**96.4**	95.5	0.1	95.9	0.1	94.3	0.7	95.0
003_cracker_box	80.5	92.7	95.5	**96.7**	0.4	96.0	0.4	94.0	0.7	96.6
004_sugar_box	90.4	**98.2**	97.5	98.1	0.2	97.6	0.2	95.7	1.6	98.2
005_tomato_soup_can	91.9	94.5	**94.6**	94.3	0.1	94.2	0.1	90.3	4.0	94.7
006_mustard_bottle	88.5	**98.6**	97.2	98.0	0.2	97.6	0.2	95.2	1.8	98.0
007_tuna_fish_can	93.8	**97.1**	96.6	96.9	0.3	96.7	0.3	95.4	0.4	96.8
008_pudding_box	87.5	**97.9**	96.5	97.3	0.5	96.3	0.5	95.1	0.8	98.2
009_gelatin_box	95.0	**98.8**	98.1	98.3	0.6	98.0	0.6	97.2	0.5	98.8
010_potted_meat_can	86.4	**92.7**	91.3	89.6	0.2	89.4	0.2	88.1	0.5	91.0
011_banana	84.7	97.1	96.6	**97.6**	0.1	97.5	0.1	95.0	0.5	97.3
019_pitcher_base	85.5	**97.8**	97.1	97.7	0.3	97.4	0.3	96.1	0.5	97.6
021_bleach_cleanser	81.0	**96.9**	95.8	95.4	0.8	93.8	0.8	94.6	0.1	96.1
024_bowl	75.7	81.0	88.2	**89.6**	3.1	90.1	3.1	88.4	0.2	89.7
025_mug	94.2	95.0	**97.1**	**97.1**	0.2	97.0	0.2	95.8	0.5	97.2
035_power_drill	71.5	**98.2**	96.0	96.7	0.3	96.4	0.3	93.9	1.1	96.6
036_wood_block	68.1	87.6	89.7	**91.8**	1.2	90.6	1.2	90.2	0.8	90.7
037_scissors	76.7	91.7	**95.2**	92.7	0.6	93.2	0.6	92.4	1.3	90.6
040_large_marker	87.9	97.2	**97.5**	**97.5**	0.3	97.0	0.3	95.7	0.5	97.1
051_large_clamp	65.9	**75.2**	72.9	71.9	1.3	72.1	1.3	69.7	0.4	74.7
052_extra_large_clamp	60.4	64.4	69.8	**71.4**	1.0	69.6	1.0	64.5	0.6	58.8
061_foam_brick	91.8	**97.2**	92.5	94.3	1.1	94.2	1.1	92.0	2.0	93.7
Average	83.9	93.0	93.1	**93.3**	0.6	92.9	0.6	91.1	0.9	92.7

4.2.2.2 YCB-Video Experiments

This subsection presents the experiments and results done on the YCB-Video data set. Once again there will be many direct comparisons with DenseFusion, because it was the best method at that time.

The results described in Table 4.4 shows the area under the accuracy-threshold curve (AUC) using the ADD-S metric (2.17).

As shown in Table 4.4, MaskedFusion obtained the best average score using 200 epochs for training. Since DenseFusion, which was the closest one, didn't state how many epochs were used to obtain those results, extra comparisons were made between MaskedFusion and DenseFusion. These comparisons consist in training both methods for only 100 epochs and then analyze the obtained values for both methods.

All experiments presented have 5 repetitions for it to be possible to compute the average error of all executions. This also includes the experiment that was done on DenseFusion with 100 training epochs. MaskedFusion achieved an average of more 0.2% in the AUC

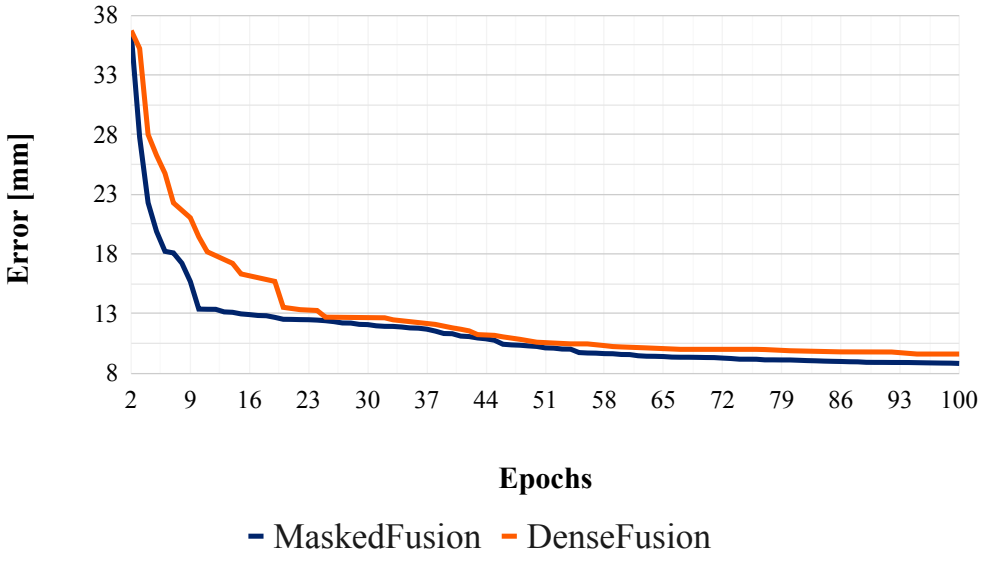
Table 4.5: Quantitative evaluation of 6D pose (percentage of ADD-S smaller than $2cm$) on the YCB-Video data set. Bold numbers are the best results in a row and underlined numbers are the best when comparing MaskedFusion with DenseFusion, both with 100 training epochs. The last column of the table is the evaluation of MaskedFusion using the masks that were generated by the first sub-task during the train and test. (*) The values presented were obtained from [WXZ+19]

	PointFusion	PoseCNN +ICP	DenseFusion	MaskedFusion		MaskedFusion		DenseFusion		Pipeline
Training Epochs	–	–	–	200		100		100		100
Objects	<2cm(*)	<2cm(*)	<2cm(*)	<2cm (Avg)	<2cm (Stdev)	<2cm (Avg)	<2cm (Stdev)	<2cm (Avg)	<2cm (Stdev)	<2cm
002_master_chef_can	99.8	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	100.0
003_cracker_box	62.6	91.6	99.5	**99.8**	0.1	99.7	0.1	99.5	0.5	99.3
004_sugar_box	95.4	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	100.0
005_tomato_soup_can	**96.9**	**96.9**	**96.9**	**96.9**	0.0	96.5	0.0	92.6	5.9	96.9
006_mustard_bottle	84.0	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	99.9	0.1	100.0
007_tuna_fish_can	99.8	**100.0**	**100.0**	99.7	0.1	99.9	0.1	100.0	0.0	100.0
008_pudding_box	96.7	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	100.0
009_gelatin_box	**100.0**	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	100.0
010_potted_meat_can	88.5	93.6	93.1	**94.2**	0.4	90.8	0.4	90.6	0.2	92.8
011_banana	70.5	99.7	**100.0**	**100.0**	0.0	100.0	0.0	99.7	0.5	99.7
019_pitcher_base	79.8	**100.0**	**100.0**	**100.0**	0.3	99.9	0.3	100.0	0.0	99.8
021_bleach_cleanser	65.0	99.4	**100.0**	99.4	3.9	94.0	3.9	100.0	0.0	99.6
024_bowl	24.1	54.9	**98.8**	95.4	5.8	95.8	5.8	95.0	4.2	91.6
025_mug	99.8	99.8	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	100.0
035_power_drill	22.8	**99.6**	98.7	99.5	0.3	99.4	0.3	97.1	1.5	99.6
036_wood_block	18.2	80.2	94.6	**100.0**	0.9	98.1	0.9	97.5	1.5	90.8
037_scissors	35.9	95.6	**100.0**	99.9	1.1	99.0	1.1	99.4	0.7	92.8
040_large_marker	80.4	99.7	**100.0**	99.9	0.1	99.7	0.1	99.8	0.1	99.4
051_large_clamp	50.0	74.9	**79.2**	78.7	0.1	77.9	0.1	75.6	1.2	80.8
052_extra_large_clamp	20.1	48.8	**76.3**	75.9	2.1	72.0	2.1	68.7	0.7	75.3
061_foam_brick	**100.0**	**100.0**	**100.0**	**100.0**	0.0	100.0	0.0	100.0	0.0	99.3
Average	74.1	93.2	96.8	**97.1**	0.7	96.3	0.7	96.0	0.8	96.1

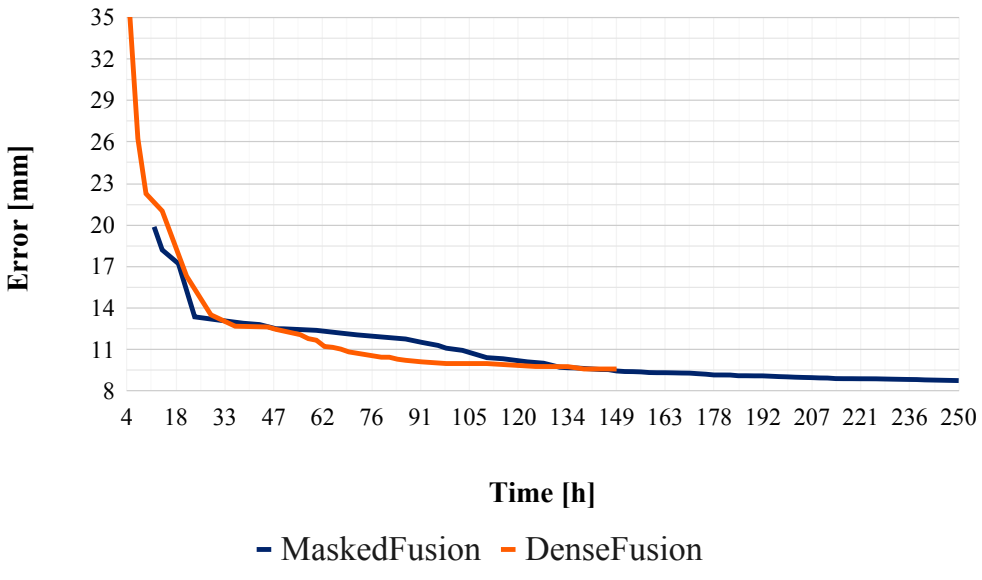
than what DenseFusion reported in their publication. But when the average of MaskedFusion is compared with DenseFusion while both using 100 training epochs, MaskedFusion achieved an an improvement of 1.8% AUC score.

On Table 4.5 it shows the percentage of ADD-S error smaller than $2cm$. This metric was introduced in DenseFusion explaining that predictions under $2cm$ are the minimum tolerance for robot manipulation. As in the AUC, DenseFusion did not state how many epochs were used for training, so for this metric, the same experiments were executed. As in the previous analysis of the results, MaskedFusion outperformed, on average, DenseFusion by obtaining an average of 0.3% better accuracy. When trained MaskedFusion and DenseFusion for 100 epochs, MaskedFusion also kept the 0.3% better accuracy in the AUC $< 2cm$ metric.

As in LineMOD data set MaskedFusion has one more network to train, additional computation over the data, and more data flowing in our method, so it took more 95 of hours than DenseFusion when comparing 100 epochs of training. MaskedFusion took 240 hours compared with 145 hours of training time for DenseFusion. Figure 4.7 presents these results considering the number of epochs and considering the time used for training.



(a) Error per epoch



(b) Error per hour

Figure 4.7: The figure contains the average error in millimeters for the YCB-Video data set. Figure (a) shows the error as a function of the training epoch, whereas figure (b) displays it as a function of training time.

4.2.2.3 Inference Pipeline Results

The inference time of MaskedFusion in the first sub-task, image segmentation, is 0.2 seconds per image. For the second sub-task, 6D pose estimation, 0.01 seconds were needed to estimate the pose per given object, and on the third sub-task, the pose refinement took 0.002 seconds. So the total inference time of MaskedFusion is 0.212 seconds from the moment of the RGB-D image capture until it has the estimated pose.

Using the masks that were generated by the first sub-task during the train MaskedFusion, achieved 92.7% in the AUC metrics (shown on the last column of Table 4.4) and 96.1% in the $< 2cm$ metric (shown on the last column of Table 4.5).

As expected, while using the predicted masks MaskedFusion had worse overall scores, since it was not using the perfect labeled masks that was provided with YCB-Video data set. This test revealed that the semantic segmentation needs to be improved in order to obtain better segmentation masks. It was noted that other methods in the literature do not show the results on the full pipeline, and consider that the method receives perfect data.

4.3 Impact of Segmentation and Color Spaces in 6D Pose Estimation

This section discusses the importance of an accurate semantic segmentation and the impact that color spaces may have on the neural networks methods that aim to solve the 6D pose estimation task. This section, starts by showing how the quality of the masks obtained from image segmentation can improve 6D pose estimation methods, like MaskedFusion [PA20b] since it relies on these masks to detect and isolate the objects present in the scene. Most methods for semantic segmentation require that each pixel has a label associated with it, such that, is possible to predict a label for every pixel of the image. Not only the class but also the boundaries of each object matter to the prediction. The output prediction also reflects the spatial-relationship of all objects present in the image. Evaluations of different color spaces are also present in this section with the objective of testing how color spaces may influence 6D pose estimation deep learning methods.

4.3.1 Impact of Image Segmentation in 6D Pose Estimation

We executed two main experiments to evaluate two hypotheses. The first consists of the belief that the use of post-processing operations over the predicted masks could improve the accuracy in the estimation of the 6D pose of an object. The second hypothesis regards

Table 4.6: Evaluation of the pre-trained weights on COCO val2017. Bold values are the best values for each metric.

Model (Backbone)	Mean IOU	Pixel-wise Accuracy
SegNet	63.2	91.3
FCN (ResNet101)	63.7	91.9
DeepLab v3 (ResNet101)	**67.4**	**92.4**

the robustness of MaskedFusion: we hypothesize that it is robust enough to deal with lower quality masks that can be produced when dealing with real-world data.

To evaluate if the first hypothesis was correct we created the first experiment that consisted of the following flow: after training the MaskedFusion method with the ground truth masks, we train each semantic segmentation neural network on the same data set, with the same train, validation and test subset. Then we feed-forward the predicted masks with and without post-processing on the test subset of the data set from each semantic segmentation method to the pre-trained MaskedFusion 6D pose and refinement network.

For the second experiment, our goal was to show how robust MaskedFusion is when using lower quality masks and how much it could improve if trained using predicted masks instead of the ground truth masks. The flow of our experiment was: training each of the semantic segmentation neural networks, and generate masks from each method for all subsets (train, validation, and test) of the data set. Then these predicted masks were used to train the MaskedFusion 6D pose and refinement networks. With this experiment, we could see the adaptation of the MaskedFusion method to different quality masks.

4.3.1.1 Results

We used three neural networks to tackle the task of semantic segmentation: SegNet [BKC17], FCN-ResNet101 [LSD15] and DeepLab v3 [CPSA17]. For SegNet we used the original implementation, for the FCN and DeepLab v3 we used ResNet101 as backbone. All of the networks used were pre-trained on COCO [LMB+14] train2017 data set and then fine-tuned.

In Table 4.6 we present the evaluation results on COCO val2017 for Mean IOU and Global Pixel-wise Accuracy for the three models. This Table give us an idea of what to expect in terms of semantic segmentation of the three tested methods.

In the first experiment with MaskedFusion, we trained it in the LineMOD data set using the ground truth masks to achieve the best possible results, since in the real-world it would

Table 4.7: Quantitative evaluation of 6D pose using the ADD metric on the LineMOD data set. Symmetric objects are presented in italic and were evaluated using ADD-S. The presented values were obtained using the data set masks to train the MaskedFusion and the evaluation used the masks generated by the semantic segmentation methods. Bold shows best results in a given row.

Objects	GT Mask	SegNet	SegNet w/ Operations	Deeplabv3 (ResNet101)	Deeplabv3 (ResNet101) w/ Operations	FCN (ResNet101)	FCN (ResNet101) w/ Operations
ape	**89.5**	81.0	80.0	84.6	87.5	82.9	86.7
bench vi.	98.1	94.2	95.1	98.1	98.1	**99.0**	98.1
camera	99.0	99.0	99.0	**100.0**	**100.0**	99.0	99.0
can	96.0	88.1	87.1	97.0	**98.0**	95.0	95.0
cat	**100.0**	97.0	97.0	**100.0**	**100.0**	**100.0**	99.0
driller	**97.0**	90.0	91.0	95.0	96.0	94.0	95.0
duck	**94.3**	87.7	90.6	91.4	93.3	88.7	90.6
eggbox	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.1
glue	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
hole p.	**98.1**	95.2	95.2	96.2	95.2	96.2	96.2
iron	**97.9**	96.9	95.9	95.9	96.9	96.9	96.9
lamp	**99.0**	97.1	96.2	97.1	96.2	98.1	**99.0**
phone	94.2	94.2	94.2	95.2	95.2	**98.1**	**98.1**
Average	**97.2**	93.9	93.9	96.2	96.6	96.0	96.3

be very difficult to obtain these precise masks. The baseline results for MaskedFusion using the ground truth masks present in the data set are shown in the second column of Table 4.7. With the weights obtained from MaskedFusion's 200 training epochs, we could evaluate the influence that image segmentation has, for the three segmentation methods.

We trained each semantic segmentation method for 50 epochs and saved the weights of the best accuracy run of the validation subset of LineMOD. For the test subset of LineMOD, we predict the mask of the input images and then we created new binary images with the predicted masks that can be used as input for MaskedFusion 6D pose estimation network during its evaluation. During the evaluation of the MaskedFusion in the test subset of the LineMOD, we used the RGB-D data provided by the data set but, for the masks used to crop the objects and to extract features from the shape of the object, we use the predicted masks.

On the top row of Table 4.7 we have the ground truth mask and the three methods tested, each with and without the filtering operations proposed by the MaskedFusion authors. The obtained results show that the proposed filtering operation can increase slightly the accuracy of the model compared with its counterpart. The MaskedFusion using DeepLab v3 masks with the filter operations median and dilate applied to its masks achieved 96.6% of average accuracy, only 0.6% below the baseline that used ground truth masks. These filter operations are described more in-depth in the section 4.2.1.

Table 4.8: Quantitative evaluation of 6D pose using the ADD metric on the LineMOD data set. Symmetric objects are presented in italic and were evaluated using ADD-S. The presented values were obtained using the masks generated by the semantic segmentation methods to train and evaluate the MaskedFusion. Bold shows best results in a given row.

Objects	GT Mask	SegNet	SegNet w/ Operations	Deeplabv3 (ResNet101)	Deeplabv3 (ResNet101) w/ Operations	FCN (ResNet101)	FCN (ResNet101) w/ Operations
ape	**89.5**	82.0	81.0	85.6	88.5	83.9	87.7
bench vi.	98.1	94.2	95.1	98.1	98.1	**99.0**	98.1
camera	99.0	99.0	99.0	**100.0**	**100.0**	99.0	99.0
can	96.0	89.1	88.1	98.0	**99.0**	96.0	96.0
cat	**100.0**	97.0	97.0	**100.0**	**100.0**	**100.0**	99.0
driller	**97.0**	90.5	91.5	95.5	96.5	94.5	95.5
duck	**94.3**	88.2	91.1	91.9	93.8	89.2	91.1
eggbox	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.1
glue	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
hole p.	**98.1**	95.7	95.7	96.7	95.7	96.7	96.7
iron	**97.9**	97.4	96.4	96.4	97.4	97.4	97.4
lamp	**99.0**	97.1	96.2	97.1	96.2	98.1	**99.0**
phone	94.2	95.2	95.2	96.2	96.2	**99.1**	**99.1**
Average	**97.2**	94.3	94.3	96.6	97.0	96.4	96.7

For the second experiment, we trained each semantic segmentation method for 50 epochs and saved the weights of the best accuracy run of the validation subset of LineMOD. With the best accuracy weights, we predicted masks for all the subsets (train, validation, test) of the LineMOD. With the predicted masks for each method, we trained MaskedFusion to check if it could adapt itself to lower quality masks. We did the same process for the methods that had the masks with the filtering operations as post-process. Table 4.8 contains the results of this experiment.

In this experiment we trained MaskedFusion for 200 epochs for each method since here the input data were changed by the different previous segmentation methods. The presented results show that the use of filtering operations after the prediction of the mask improves the accuracy of the object estimated pose, and we showed that MaskedFusion can adapt itself to lower quality masks since in this experiment all the masks used were provided by semantic segmentation methods. The use of the masks produced by the method DeepLab v3 achieved again the best average accuracy for this problem resulting in an accuracy 0.2% below the ground truth baseline.

4.3.2 Impact of Different Color Spaces in 6D Pose Estimation

In our color spaces experiments, we used DenseFusion [WXZ+19] and MaskedFusion [PA20b], but we did not use the first sub-task (semantic segmentation) of the MaskedFusion. Our primary goal is to report the impact of the different color spaces and/or channels in the 6D pose estimation. Since MaskedFusion is a modular framework, it was effortless to remove the semantic segmentation sub-task and use the ground truth masks to make

Table 4.9: Quantitative evaluation of 6D pose using the ADD metric on the LineMOD data set. Symmetric objects are presented in italic and were evaluated using ADD-S. Bold numbers are the best in a row for MaskedFusion and underline bold numbers are the best in a row for DenseFusion both methods were trained for 200 epochs.

	MaskedFusion [PA20b]						DenseFusion [WXZ+19]					
Objects	RGB	HSV	Gray	H	S	V	RGB	HSV	Gray	H	S	V
ape	89.5	**97.1**	86.7	67.6	34.3	82.9	92.3	**92.8**	85.8	70.4	37.1	85.6
bench vi.	98.1	99.0	**100.0**	88.3	89.3	99.0	93.2	93.9	90.2	83.5	84.5	**94.2**
camera	**99.0**	98.0	98.0	87.3	75.5	97.1	94.4	**95.8**	92.4	82.6	70.9	92.4
can	96.0	**98.0**	97.0	80.2	91.1	93.1	93.1	**93.7**	92.1	77.3	88.1	90.1
cat	**100.0**	97.0	95.0	81.0	86.0	97.0	**96.5**	96.4	94.6	77.5	82.5	93.5
driller	97.0	**99.0**	95.0	91.0	88.0	94.0	87.0	**87.4**	85.0	81.0	78.0	84.0
duck	94.3	**96.2**	93.4	51.9	35.8	88.7	**92.3**	92.0	90.1	49.8	33.8	86.6
eggbox	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**99.8**	**99.8**	95.8	**99.8**	**99.8**	**99.8**
glue	**100.0**	**100.0**	99.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	98.1	**100.0**	**100.0**	**100.0**
hole p.	98.1	**99.0**	95.2	89.5	74.3	**99.0**	92.1	92.4	89.8	83.5	68.3	**93.1**
iron	97.9	95.9	97.9	94.8	91.8	**99.0**	97.0	97.5	95.4	93.9	90.8	**98.0**
lamp	99.0	98.1	**100.0**	95.2	97.1	**100.0**	95.3	95.8	91.8	91.5	93.4	**96.3**
phone	94.2	**100.0**	99.0	93.3	94.2	99.0	92.8	92.7	89.4	91.8	92.8	**97.6**
Average	97.2	**98.3**	96.6	86.2	81.3	96.1	94.3	**94.6**	91.6	83.3	78.5	93.2

the operations for the detection, crop, and background removal. This also enables us to have a direct comparison between DenseFusion and MaskedFusion and know the possible improvements that can be achieved in both methods by using different color spaces.

To perform our tests, we choose to compare the HSV color space and each of its channels with the RGB color space. Thus meaning that we used the RGB, HSV, Grayscale, H (Hue), S (Saturation), and V (Value). We evaluated the DenseFusion [WXZ+19] and MaskedFusion [PA20b] by training both from randomly initialized weights for 200 epochs in the LineMOD data set for all the aforementioned color spaces and channels and 100 epochs in the YCB-Video data set.

4.3.2.1 Results

In Table 4.9 we present the quantitative evaluation for the LineMOD data set of the two methods that we used to estimate the object's 6D pose. On average both methods had less pose error using the HSV color space. Since LineMOD is a data set where the objects have less texture and is colorful, using a different color space as HSV, improved slightly the accuracy.

The YCB-Video data set quantitative evaluation is presented in Table 4.10 comparing each color space for both methods used in this experience. In the YCB-Video data set, the objects have more texture and they are less colorful thus not enabling the HSV to have a big advantage over the other color spaces. In the MaskedFusion experiments, we obtained the same average accuracy in the HSV and Gray color space. The color space gray has higher scores in objects that have more texture or objects that are black, comparing to the other color spaces.

Table 4.10: Quantitative evaluation of 6D pose (area under the ADD-S curve(AUC)) on the YCB-Video data set. Bold numbers are the best in a row for MaskedFusion and underline bold numbers are the best in a row for DenseFusion both methods were trained for 100 epochs.

	MaskedFusion [PA20b]						DenseFusion [WXZ+19]					
Objects	RGB	HSV	Gray	H	S	V	RGB	HSV	Gray	H	S	V
002_master_chef_can	95.9	**96.2**	95.9	95.6	95.6	96.1	94.3	**94.6**	94.3	94.0	94.0	94.5
003_cracker_box	96.0	96.5	96.2	95.6	**96.6**	95.3	94.0	94.5	94.2	93.5	**94.6**	93.3
004_sugar_box	97.6	97.8	97.7	97.6	**97.9**	97.7	95.7	95.9	95.6	95.7	**95.9**	95.8
005_tomato_soup_can	94.2	**94.3**	94.3	93.8	94.1	94.1	90.3	**90.5**	90.4	89.9	90.2	90.1
006_mustard_bottle	97.6	97.9	96.9	**98.1**	97.0	96.3	95.2	95.5	95.0	**95.6**	94.5	93.9
007_tuna_fish_can	96.7	97.0	**97.1**	96.7	97.0	96.5	95.4	95.5	95.4	95.4	**95.7**	95.3
008_pudding_box	96.3	96.8	96.5	94.6	97.0	**97.2**	95.1	95.9	94.9	93.4	95.9	**96.0**
009_gelatin_box	98.0	97.9	**98.2**	98.1	98.2	97.5	97.2	97.6	**97.7**	97.3	97.4	96.7
010_potted_meat_can	89.4	89.4	**89.5**	88.8	88.9	89.0	88.1	**88.3**	**88.3**	87.5	87.5	87.7
011_banana	**97.5**	**97.5**	97.3	**97.5**	97.2	96.4	95.0	**95.3**	94.8	95.0	94.7	93.9
019_pitcher_base	97.4	97.7	98.1	97.9	97.7	97.6	96.1	96.4	96.4	**96.6**	96.4	96.2
021_bleach_cleanser	93.8	**96.4**	95.1	93.8	95.0	94.0	94.6	94.9	94.5	94.6	**95.8**	94.8
024_bowl	**90.1**	87.8	90.0	90.0	88.6	88.0	88.4	**89.1**	88.0	88.2	86.8	86.2
025_mug	97.0	97.3	97.3	**97.6**	97.5	97.2	95.8	95.8	93.2	**96.3**	96.3	96.0
035_power_drill	96.4	**96.9**	96.5	95.9	94.5	96.1	93.9	**94.2**	93.8	93.3	92.0	93.6
036_wood_block	90.6	91.7	**92.4**	91.7	89.7	91.2	90.2	90.9	90.7	**91.3**	89.2	90.8
037_scissors	**93.2**	91.1	91.0	92.0	90.9	92.4	92.4	**93.0**	92.3	91.2	90.1	91.6
040_large_marker	97.0	97.3	**97.5**	96.6	97.2	96.9	95.7	95.7	95.8	95.3	**95.9**	95.6
051_large_clamp	72.1	72.0	**72.3**	72.0	71.3	71.8	69.7	70.2	**70.3**	69.6	68.9	69.4
052_extra_large_clamp	69.6	**72.2**	**72.2**	71.0	71.5	72.1	64.5	65.0	65.1	65.9	66.5	**67.0**
061_foam_brick	94.2	**95.2**	94.6	93.7	92.5	94.5	92.0	**92.7**	92.1	91.5	90.3	92.3
Average	92.9	**93.2**	**93.2**	92.8	92.7	92.8	91.1	**91.5**	91.1	91.0	90.9	91.0

During inference, we took an average of 0.014 seconds to estimate the 6D pose of an object. Our experiments took on average more 0.002 seconds to estimate the 6D pose of an object comparing its execution time with the RGB color space that did not need any color space conversion.

4.4 MPF6D: Masked Pyramid Fusion 6D Pose Estimation

MPF6D: Masked Pyramid Fusion 6D Pose Estimation is our latest method that tends to solve the 6D pose estimation problem. It leverages pyramid neural networks techniques to extract and fuse features at multiple resolutions. With the extraction of features at multiple resolutions the model builds a better representation of the input data. Fusing the most significant features enabled the model to keep the most significant features of each level, thus achieving good results while keeping the low inference time of our previous methods.

4.4.1 MPF6D

MPF6D has influences from pyramid neural networks. It leverages this type of architecture during the feature extraction and feature fusion to have multiple features that combine low-resolution features, semantically strong features, and high-resolution features. With this type of architecture, it was possible to create an accurate and fast method to

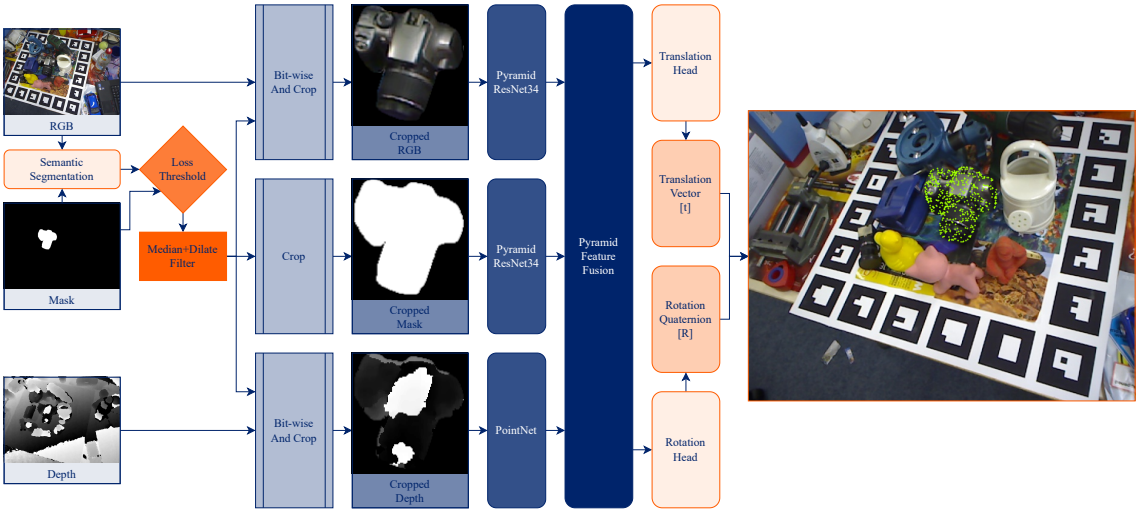


Figure 4.8: Representation of the data-flow thought MPF6D without the Pose Refinement optional step. Note that this flow is for the training phase. At inference time, the mask is generated by our method using the semantic segmentation head.

estimate the 6D pose of objects.

Our architecture has five steps from the input data until we have the estimated pose. The five steps are Semantic Segmentation, Feature Extraction, Feature Fusion, Pose Estimation, and Pose Refinement. All five steps are trained simultaneously.

In Figure 4.8, we have the overview diagram representing the data flow through MPF6D, except for the optional Pose Refinement step. The flow starts on the top of the image with the RGB, Mask, and Depth images as input. Note that this flow is for the training phase. At inference time, the mask is generated using the semantic segmentation head. Figure 4.9 represents the pyramid scheme ResNet34 (we named it Pyramid ResNet34) and the upscale processes to have features from different scales in a pixel-wise form. In Figure 4.10, we show the fusion of the multiple extracted features that are used in the estimation heads (Translation Head, Rotation Head).

In the first step, Semantic Segmentation, we used the DeepLabV3 [CPSA17] architecture as a head of our method to detect, classify and generate the mask of the known objects presented in the scene. This head is trained using the same loss function that was proposed by the original authors in [CPSA17]. The training is done simultaneously as the rest of the method. This train was possible due to the integration of this head into our neural network. After the prediction of the mask, we apply a technique introduced in MaskedFusion [PA20b] where, first, we use a median filter to smooth the mask image with a kernel size of 3×3, and then we dilate the mask with a 5×5 kernel such that, if the mask has some minor boundary segmentation error, this operation helps to correct

it or complete any misclassified pixel in the middle of the object. Since the 6D pose estimation step of our neural network requires an isolated object as input, we use the mask of the classified object to crop that object. For that, we needed to add a mechanism that would enable the rest of the neural network to be trained efficiently instead of waiting for the segmentation head to be accurate. This mechanism is only used in the initial training epochs. The mechanism consists in using the ground truth mask in the 6D pose estimation step while the semantic segmentation head does not achieve a low and stable loss. We use the ground truth masks until the threshold mechanism detects that the loss of the segmentation head is stabilizing. Then we start to use masks produced by the segmentation head into the next steps. We apply and analyze the loss threshold in the validation subset. We use a *bit-wise AND* between the mask and the RGB image and the mask and depth image to only have the pixels that have the object in it. Then we do a rectangular crop of the object from the resultant images. The crop enables us to have a smaller size tensor than the whole image as an input tensor, where most of the pixels were black due to our *bit-wise AND* operation to remove the background.

For the Feature Extraction step of the RGB and mask data, we use a ResNet34 architecture in a pyramid-like architecture (Figure 4.9) where we stacked two of the original ResNet34 architectures to have multi-scale features. This ResNet34 has at the end upscale layers. This type of layer is similar to the ones presented in PSPNet [ZSQ+17]. They consist of convolutional layers mixed with upsampling layers that enable us to assign features for each original pixel of the object (pixel-wise features). We use the features produced by the first ResNet34 as input to the next ResNet34, and we upscale these features to the original object size after the first ResNet34, and after the second ResNet34, then we fuse both of the upscaled features. The fusion is made with a concatenation of the features tensors and the output of two convolutional layers. These fused features for the RGB data and mask will be used in the pyramid feature fusion.

For the depth data, we convert the cropped depth image into a point cloud, and then we use a PointNet architecture to extract features from the generated point cloud.

The neural network responsible for the feature extraction generates features corresponding to each different data type. For the feature fusion step (Figure 4.10), we use a pyramid-like architecture with the intent of having multi-dimensional features with different scales that are then upscaled to the original object image size. The pyramid feature fusion has three different resolutions. With this technique, which was previously used during the feature extraction of RGB and mask data, we can have the most significant features of the object while keeping the original features that represent the object's size, geometry, and pixel-wise multi-feature. With all the fused features, we can then use the neural network heads to estimate the different values for the object's position and its rotation. We

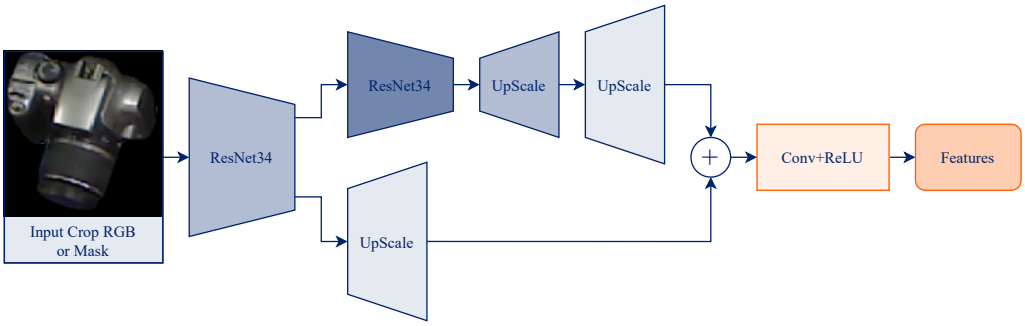


Figure 4.9: In-depth representation of Pyramid ResNet34. This system is used with both RGB and Mask images to extract features, with only one change between them: the first layer for the mask receives only one channel instead of three.

use two regression heads, one for estimating the translation vector of the object and the other to estimate the quaternion that corresponds to the object's rotation. After having this preliminary 6D pose of the object, we could use other methods (ICP or DenseFusion refinement) to refine the 6D pose estimation. However, we improved the DenseFusion refinement neural network to use it in our refinement step. We choose to improve upon DenseFusion since their refinement neural network can be used during the inference time without too much computation time. We added two extra layers to use the same type of pyramid architecture used before to maintain the original scale of the features and have deeper features.

To train MPF6D, we use the following loss function (4.3) where we calculate the error between M randomly sampled points and the ground truth object pose:

$$\mathcal{L}_i^p = \frac{1}{M} \sum_j \left\| (Rx_j + t) - (\hat{R}_i x_j + \hat{t}_i) \right\| \quad (4.3)$$

where, x_j denotes the j^{th} point of the M randomly selected 3D points from the object's 3D model, $p = [R|t]$ is the ground truth pose, R is the rotation matrix of the object and t is the translation vector. The estimated pose from MPF6D is represented by $\hat{p}_i = [\hat{R}_i|\hat{t}_i]$ where $\hat{R}$ denotes the predicted rotation and $\hat{t}$ the predicted translation.

All these techniques, in conjunction, enable us to have an accurate 6D pose estimation while keeping the inference time as low as possible to enable our method to be used in real-world applications.

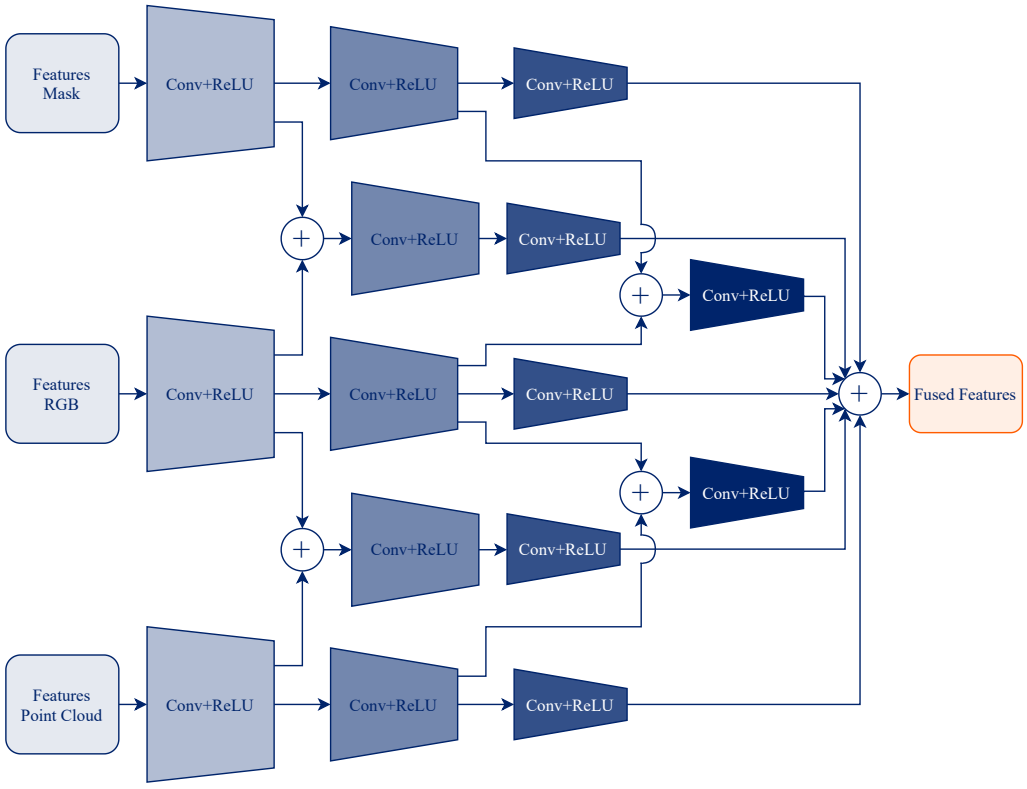


Figure 4.10: Pyramid Fusion is the architecture that fuses extracted features from the different data types (RGB, Mask, Depth/Point Cloud).

4.4.2 Experiments

As in previous sections, we used two data sets, LineMOD and YCB-Video, as well as the presented evaluation metrics in section 2.7.4 to evaluate the performance of the proposed method.

4.4.2.1 Results: LineMOD

The cluttered sceneries, texture-less objects, and lighting fluctuations, as previously indicated, are the main challenges of this data set. Our technique achieved less pose error under these complex settings than all other methods.

As presented in Table 4.11, with a direct comparison of our method and PVN3D, we improved by 0.3%. This value might not be seen as much improvement, but since our method, PVN3D, and even MaskedFusion are close to the zero error mark, all slight improvements are hard to get. Extracting features from different resolutions enable a more accurate pose estimation, independent of the camera angle and object distance. MPF6D achieved the best accuracy in 9 objects out of 13 in its best execution. The values presented in the second column of Table 4.11 correspond to the best run. Each run was trained from

Table 4.11: Quantitative evaluation of 6D pose using the ADD (2.16) metric on the LineMOD data set. Symmetric objects are presented in italic and were evaluated using ADD-S (2.17). Bold shows best results in a given row.

Objects	MPF6D Avg (Stdev)	MPF6D*	PVN3D	MaskedFusion	DenseFusion	PointFusion	SSD-6D+ICP	Implicit+ICP
ape	98.9 (0.3)	**99.2**	97.3	91.4	92.3	70.4	65.0	20.6
bench vi.	99.7 (0.3)	99.5	**99.7**	99.0	93.2	80.7	80.0	64.3
camera	100.0 (0.0)	**100.0**	99.6	99.0	94.4	60.8	78.0	63.2
can	98.9 (0.3)	99.2	99.5	**100.0**	93.1	61.1	86.0	76.1
cat	99.7 (0.3)	**100.0**	99.8	**100.0**	96.5	79.1	70.0	72.0
driller	99.8 (0.1)	**99.8**	99.3	97.0	87.0	47.3	73.0	41.6
duck	99.1 (0.2)	**99.3**	98.2	92.5	92.3	63.0	66.0	32.4
eggbox	100.0 (0.0)	**100.0**	99.8	99.1	99.8	99.9	**100.0**	98.6
glue	100.0 (0.0)	**100.0**	**100.0**	**100.0**	**100.0**	99.3	**100.0**	96.4
hole p.	99.8 (0.1)	99.8	99.9	**100.0**	92.1	71.8	49.0	49.9
iron	99.7 (0.0)	**99.8**	99.7	96.9	97.0	83.2	78.0	63.1
lamp	99.6 (0.3)	**99.8**	**99.8**	98.1	95.3	62.3	73.0	91.7
phone	99.3 (0.2)	99.1	**99.5**	99.0	92.8	78.8	79.0	71.0
Average	99.6 (0.1)	**99.7**	99.4	97.8	94.3	73.7	76.7	64.7

(*) Best of three repetitions.

scratch, where all the weights were initialized randomly. The first column shows the average values and standard deviation for all three runs. We can see that even the average of our three repetitions presents better results than any competing approaches.

4.4.2.2 Results: YCB-Video

In Table 4.12, we present the quantitative evaluation using the area under the ADD-S (2.17) curve (AUC). Our method outperforms all previous methods. Comparing it with PVN3D, we had 1.96% more area under the curve. In the YCB-Video data set, our method had the best accuracy for 19 objects out of 21. The two objects where we lose for the PVN3D are the same object (clamp) but with different sizes. As with LineMOD, the average of our three repetitions has a better average result than any of the other methods.

Figure 4.11 shows two examples of poor performance of object pose estimation on the left and two good performance examples on the right. The two right examples are also good examples of the MPF6D handling object occlusions without losing accuracy.

4.4.2.3 Inference

MPF6D can be used in real-time applications since it can infer the 6D pose of an object in 0.12 seconds so that it can execute at eight frames per second. This time was measured from the instant the data (RGB-D) was fed to the method until it produced the 6D pose estimation (translation vector and quaternion with the rotation representation). We need an extra 0.02 seconds to have the output as a translation vector and a rotation matrix. PVN3D only reports the inference time on the LineMOD data set, but in the YCB-Video

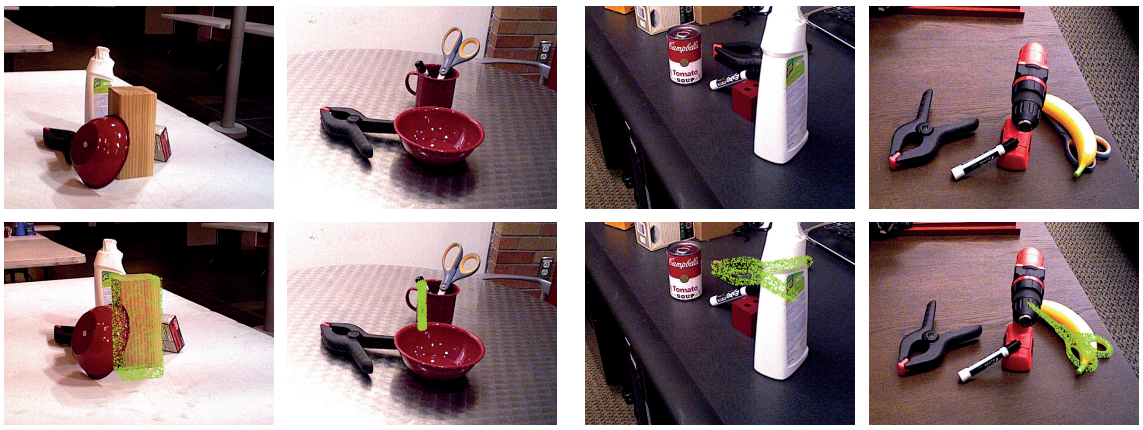


Figure 4.11: MPF6D object pose estimation examples. The green dots represent keypoints of the object pose estimation projected onto the RGB image. The top row contains the input RGB images and the bottom row the predicted object poses. The left two columns contain two examples of poor performance and the right two columns of good performance under heavy occlusion.

Table 4.12: Quantitative evaluation of 6D pose (area under the ADD-S (2.17) curve (AUC)) on the YCB-Video data set. Bold numbers are the best in a row.

Objects	MPF6D Avg (Stdev)	MPF6D*	PVN3D+ICP	MaskedFusion	DenseFusion	PointFusion	PoseCNN+ICP
002_master_chef_can	99.69 (0.22)	**99.44**	95.20	96.91	96.40	90.90	95.80
003_cracker_box	99.60 (0.39)	**99.75**	94.40	96.55	95.50	80.50	92.70
004_sugar_box	99.49 (0.21)	**99.47**	97.90	98.81	97.50	90.40	98.20
005_tomato_soup_can	99.12 (0.28)	**99.08**	95.90	95.64	94.60	91.90	94.50
006_mustard_bottle	99.51 (0.37)	**99.61**	98.30	98.13	97.20	88.50	98.60
007_tuna_fish_can	99.50 (0.43)	**99.52**	96.70	97.31	96.60	93.80	97.10
008_pudding_box	99.68 (0.40)	**99.89**	98.20	97.02	96.50	87.50	97.90
009_gelatin_box	99.45 (0.24)	**99.41**	98.80	98.69	98.10	95.00	98.80
010_potted_meat_can	97.47 (0.23)	**97.64**	93.80	94.57	91.30	86.40	92.70
011_banana	99.47 (0.41)	**99.50**	98.20	98.10	96.60	84.70	97.10
019_pitcher_base	99.39 (0.46)	**98.87**	97.60	97.06	97.10	85.50	97.80
021_bleach_cleanser	99.02 (0.47)	**98.99**	97.20	96.53	95.80	81.00	96.90
024_bowl	98.46 (0.20)	**98.51**	92.80	97.55	88.20	75.70	81.00
025_mug	99.02 (0.45)	**98.84**	97.70	97.48	97.10	94.20	95.00
035_power_drill	99.15 (0.54)	**99.54**	97.10	97.26	96.00	71.50	98.20
036_wood_block	95.81 (0.15)	**95.98**	91.10	95.42	89.70	68.10	87.60
037_scissors	97.13 (0.42)	**97.10**	95.00	95.93	95.20	76.70	91.70
040_large_marker	99.45 (0.40)	**99.02**	98.10	97.55	97.50	87.90	97.20
051_large_clamp	88.97 (5.09)	93.13	**95.60**	89.40	72.90	65.90	75.20
052_extra_large_clamp	82.73 (5.93)	87.11	**90.50**	84.04	69.80	60.40	64.40
061_foam_brick	98.85 (0.11)	**98.83**	98.20	95.29	92.50	91.80	97.20
Average	97.66 (0.37)	**98.06**	96.10	95.96	93.20	83.90	93.00

(*) Best of three repetitions.

Table 4.13: Quantitative inference time. The values presented in the table were measured in seconds.

Methods	Segmentation	6D Pose	Pose Refinement	Overall
DenseFusion	0.03	0.02	0.01	0.06
MPF6D	-	-	-	0.12
PVN3D	-	0.02	- (*)	> 0.17 (*)
MaskedFusion	0.2	0.01	0.002	0.212
PoseCNN+ICP	0.03	0.17	10.4	10.6

(*) Pose Refinement time not reported by the authors.

Table 4.14: Ablation studies (using the ADD (2.16)) on the LineMOD data set. Ablation 1, removing the Pyramid ResNet34. Ablation 2, lower depth Pyramid Fusion. Ablation 3, removing Pyramid Fusion.

Objects	MPF6D Architecture	Ablation 1	Ablation 2	Ablation 3
ape	90.3	75.2	79.5	75.5
bench vi.	90.5	73.1	79.8	75.3
camera	96.7	78.7	85.4	82.0
can	90.3	79.9	78.8	77.6
cat	96.2	77.4	82.0	84.1
driller	90.9	77.9	81.0	80.1
duck	90.3	81.1	77.6	77.0
eggbox	96.9	78.4	87.0	82.4
glue	96.8	82.1	89.9	83.4
hole p.	90.9	80.9	83.0	78.8
iron	90.8	76.8	81.4	78.2
lamp	90.8	76.0	83.4	75.2
phone	90.2	77.9	82.2	77.9
Average	92.4	78.1	82.4	79.0

the authors reported the results where they used the ICP to refine the 6D pose. Using the ICP algorithm improves the overall 6D pose estimation, but usually, this algorithm has high computation costs. It is possible to see that PoseCNN used the ICP refinement, and just for the refinement, it spent 10.4 seconds.

In Table 4.13, we present the quantitative comparison of values measured in seconds of methods that reported their inference times. The fastest method for inference is DenseFusion, but in terms of accuracy, we should compare the two best methods, ours and PVN3D. Comparing these two, MPF6D has 0.05 seconds faster inference time than PVN3D even while using pose refinement.

4.4.3 Ablation Studies

For the ablation studies in our method, we performed three extra experiments in both data sets (LineMOD and YCB-Video). For these experiments, we trained our method for 50 epochs in the LineMOD and YCB-Video data sets, and we evaluated the inference output on the test subset of each data set. All the experiment results executed in the LineMOD data set are shown in Table 4.14 and the obtained results for the YCB-Video are shown in Table 4.15. Ablation study one tests the impact of removing the pyramid from the feature extraction step (Pyramid ResNet34), thus meaning that we only used a single ResNet34 and one upsample layer and then proceeds to the fusion layers of the neural network. With this experiment, it is possible to analyze the influence of our pyramid architecture on the MPF6D backbone feature extraction for the mask and RGB data. Without the Pyramid

Table 4.15: Ablation studies (using area under the ADD-S (2.17) curve (AUC)) on the YCB-Video data set. Ablation 1, removing the Pyramid ResNet34. Ablation 2, lower depth Pyramid Fusion. Ablation 3, removing Pyramid Fusion.

Objects	MPF6D Architecture	Ablation 1	Ablation 2	Ablation 3
002_master_chef_can	88.38	75.78	79.01	73.98
003_cracker_box	88.66	75.79	78.91	74.38
004_sugar_box	88.46	75.74	79.56	75.09
005_tomato_soup_can	88.32	75.62	79.00	73.32
006_mustard_bottle	88.49	75.77	79.04	73.91
007_tuna_fish_can	88.11	74.94	78.69	73.37
008_pudding_box	88.15	75.13	78.37	73.17
009_gelatin_box	88.68	75.82	79.17	74.45
010_potted_meat_can	85.91	73.32	76.34	72.53
011_banana	88.83	76.37	80.37	74.22
019_pitcher_base	88.61	75.84	78.80	73.53
021_bleach_cleanser	88.13	75.35	79.02	73.60
024_bowl	87.55	74.48	78.33	73.20
025_mug	88.11	75.54	78.70	74.40
035_power_drill	87.70	75.36	78.58	73.01
036_wood_block	84.60	72.02	75.92	70.41
037_scissors	86.45	73.63	77.32	72.56
040_large_marker	88.15	75.45	78.34	73.93
051_large_clamp	74.08	63.34	65.89	62.80
052_extra_large_clamp	67.62	57.19	61.03	56.01
061_foam_brick	87.55	74.51	78.40	72.72
Average	86.22	73.67	77.09	72.12

ResNet34, the method obtained a 15% increase in the error rate when compared to the original MPF6D architecture, thus meaning that having features extracted with multiple resolutions will improve the object 6D pose estimation. The second ablation study focuses on the impact of the Pyramid Fusion depth. This study removes one depth level of the Pyramid Fusion, thus enabling us to study if, with a lower depth, we could achieve the same results. The obtained results had around 10% more error overall. The third ablation study evaluates the impact of removing the Pyramid Fusion from the original architecture. We replaced the Pyramid Fusion with a simple concatenation of the multiple features, a convolution layer, and a ReLU activation function. This third experiment showed that fusing multiple feature resolutions without using the pyramid approach increases the overall error by 15%.

4.5 Conclusion

Achieving a robust estimation of the 6D pose of objects captured in the real-world is an open challenge, due to infinite object shapes, material types and color combinations

combined with multiple light conditions making this area of computer vision field hard to implement in the real-world. To tackle this challenge we proposed 4 new methods.

In the first section, we tried to improve the DenseFusion architecture without considering the pose refinement phase. We found that 30 epochs were enough to train the original DenseFusion without the refinement. With these 30 epochs, it is possible to achieve good results in terms of pose estimation while using as little time as possible to train the method. The pose refinement phase helps the method to achieve higher accuracy with the drawback of needing more time to train. We also showed that using global descriptors instead of the PointNet algorithm to extract features from the point clouds can be an advantage due to the 19% less time needed to train this type of architecture with the cost of less than $1mm$ of error. In the real-world, this might be advantageous because when a new object is needed, the method needs to be fine-tuned or retrained for these new objects, and it has high computation and time costs.

MaskedFusion shown in the second section, improved upon previous methods in this area using objects' masks retrieved during the semantic segmentation, to identify and localize the object in the RGB image. An error bellow $6mm$ in LineMOD was achieved with only 100 training epochs, and in YCB-Video, which is a challenging data set, it obtained an AUC score of 93.3% and 97.1% in the $< 2cm$ metric. The masks are used to remove non-relevant data from the input of the neural network and serve as an additional data to extract features from the shape of the object. MaskedFusion has low inference time compared with previous methods but at the cost of an increase in training time. Increasing the training time can sometimes be disregarded. Since MaskedFusion still beats the previous methods, if the training time stops after a fixed number of hours instead of a fixed number of epochs, such as with LineMOD data and during most of YCB-Video training. The use of masks can be seen as a type of attention mechanism, forcing the network to process only the region highlighted by the mask.

In the third section, we presented multiple experiments that were done to further analyze two important factors that influence the 6D pose estimation problem: the quality of segmentation masks for MaskedFusion and the color space used for representing the captured images for MaskedFusion and DenseFusion. We concluded that, for the two 6D pose estimation methods that we compared, using different color spaces can improve the accuracy of the estimated pose. For MaskedFusion, that uses semantic segmentation masks of the objects to remove the background and information not related with the object, using state-of-the-art methods in the semantic segmentation with post-processing filters to smooth the obtained masks can improve the accuracy of the 6D pose estimated.

In terms of using different color spaces for the 6D pose estimation, we learned that if we are estimating poses of colorful objects, the HSV color space could improve these methods because the color features might be more relevant to the neural networks. For objects with significant textures, for example, objects packed in boxes with labels and drawings in them, the HSV color space did not perform as well as in other cases, and we concluded that for these types of objects we should use the Gray or the RGB color space.

Overall, using the HSV color space, we improved MaskedFusion in 1.1% and DenseFusion 0.3% in the LineMOD data set and 0.3% for the MaskedFusion and 0.4% for the DenseFusion in the YCB-Video data set. These are small improvements but these methods already have high accuracy in these data sets, so it is very difficult to obtain large significant gains.

In the last section, we proposed a new method that consists of a single feed-forward network that can do the complete inference from data to 6D pose estimation. Our method can be used in real-time taking only 0.12 seconds to retrieve an accurate 6D pose estimation of a known object present in the scene. Our method has the best overall performance in both used data sets (LineMOD and YCB-Video). In the LineMOD data set, we achieved 99.7% of accuracy, having 0.3% better accuracy than the second-best method PVN3D. In the YCB-Video data set, we achieved 98.06% area under the ADD-S curve which is 1.96% better than PVN3D. We performed ablation studies to clarify the impact of the three main architecture components on the overall error rates and found that the removal of these components accounts for a similar error increase on both used data sets, indicating that their benefits are not dependent of the particular data used.

Overtime we propose new approaches to this open challenge and we could improve the area with the developed methods, thus enabling an overall higher accuracy in the most used data sets and having good inference times that enables the use of the methods in real-time.

Chapter 5

Robot Grasping Approaches

In the present chapter, two approaches for robot grasping are presented and discussed. These solutions were applied in real-world robots in order to execute pick-and-place tasks in unconstrained environments. The first approach is a pipeline that uses object 6D pose estimation to recognize and detect where the object is located, and based on the object's pose, a grasping point is chosen. For the second approach a solution that optimizes the usual pipelines for robot grasping is proposed. This last solution was developed and implemented in order to have a simpler pipeline that uses less computational resources, thus enabling it to run on lower end hardware and avoiding long training times for multiple deep learning methods.

Foreword on Author Contributions

The results of this work have been disseminated in the form of two papers.

- [LPA19] V. Lopes, N. Pereira, and L. A. Alexandre, "Robot workspace monitoring using a blockchain-based 3d vision approach," in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2812–2820, 2019.
- [PA22] N. Pereira, L. A. Alexandre, "An Efficient Pick-and-Place Pipeline based on Quantized Segmentation", International Conference on Machine Learning, Control, and Robotics, (MLCR 2022), October 29-31, 2022.

5.1 Real-world Application

With the objective in hand to grasp objects in unconstrained environments, a deep learning solution was developed to receive data from an RGB-D camera and output the grasping position for a robotic arm to be able to move to that position and pick the intended object.

For this solution multiple research projects needed to be combined. Figure 5.1, represents the object grasping pipeline that starts with the data capture and ends with the grasping coordinates.

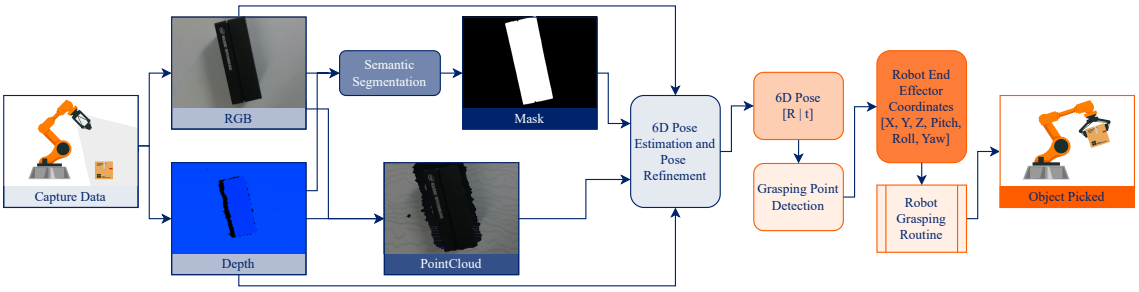


Figure 5.1: The diagram shows the processes and data flow involved in the object grasping pipeline, which starts by capturing data with an RGB-D camera and computes everything needed until the robot has grasped the intended object.

To capture data, an RGB-D camera is required. With the RGB and Depth images, a segmentation of the known objects present in the scene is made. The segmentation classifies every pixel of the RGB image and generates the masks for each object. After the segmentation, a median filter and a dilate filter are applied to the masks. The median filter is used to smooth the mask with a kernel size of 3×3, and then the dilate filter is applied to the mask with a 5×5 kernel such that if the mask has some minor boundary segmentation error, this operation helps by including more pixels around the segmented object. With the applied filters on the masks a *bit-wise and* operation is used between the segmented mask for each object and both RGB and depth images to isolate only one object from the scene to then perform a crop that will only keep the selected object in it. With all the data processed and the objects presented in the scene having been segmented, the 6D pose estimation method is ready to be used for each object and estimate its 6D pose. The 6D pose estimation method requires an RGB image, a point cloud, and a mask. It is possible to convert the cropped depth image into a point cloud. This conversion is done to then send the cropped RGB image, the Point Cloud, and the mask to the neural network. After the forwarding process through the neural network, the 6D pose is estimated, thus meaning that the position of the object and rotation are known regarding the camera viewpoint. MaskedFusion [PA20b] and MPF6D [PA23] articles show more in-depth the neural network architectures that are used to solve the 6D pose estimation task. Incorporating both pose estimation techniques into our pipeline is feasible. After conducting experiments on both methods, we observed no significant differences in the pipeline's performance, despite MaskedFusion's lower accuracy.

The output 6D pose of the object is then used in the method that will give the ideal grasping point for each object. This method requires a calibration between the robot coordinates space and the camera viewpoint space, and then the center of the object obtained from the 6D pose estimation is converted into the robot coordinates. To convert the coordinate spaces the Kabsch [Kab76] algorithm is used. Having the robot coordinates it is possible to send the gripper to that position and execute the object grasp.

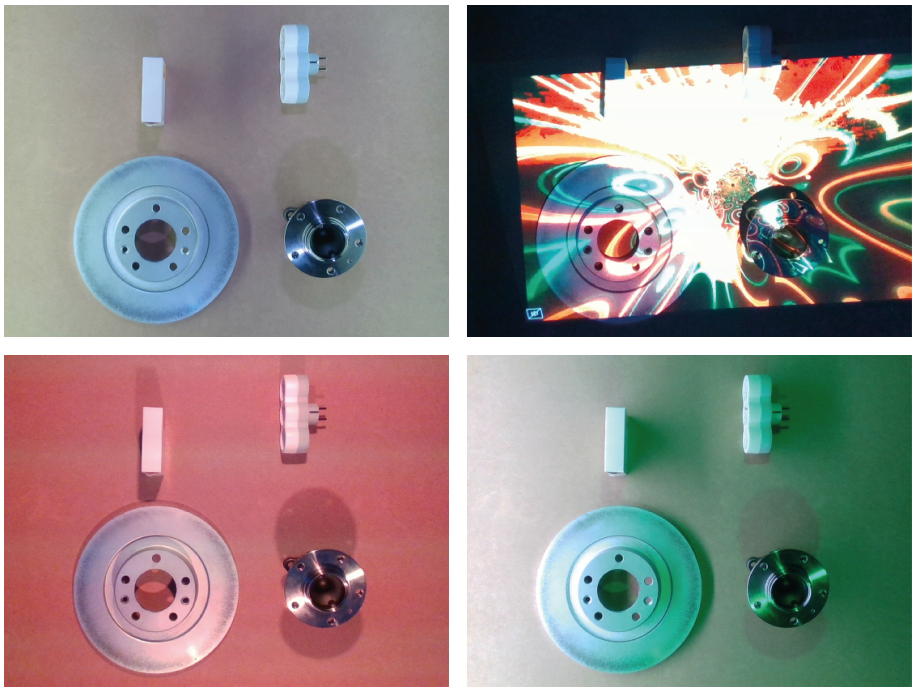


Figure 5.2: Example of RGB images present in the real-world data set. Data set used to test the methods developed during this thesis.

5.1.1 Our Data Set

To test our frameworks in a real-world environment, and then control a real robot to pick-and-place objects in a unconstrained environment, a data set needed to be created. The creation of the data set comes from two requirements, first the detection and pose estimation of two specific objects, car bearing and disk. These two objects are present in a project that this work is involved, and the project required that a robot would pick these two objects in an unconstrained environment and give them to an human operator. The second requirement was that the objects present in the LineMOD and YCB data sets were hard to find and buy in Portugal, thus we needed to find alternative objects. For this data set four classes with object variations in some classes were chosen. The classes present in it are, bearing (car bearing), disk (car disk), adapter (outlet plug adapter), box (cardboard box). For the adapter and box class the data set has variations of the objects present in them, thus having different colors and textures, for example the adapter class has two variations one white with a gray texture other pure black. Figure 5.2 show some RGB images present in the data set.

To acquire the data set two Intel RealSense Depth Camera, the model D415 and D435, were used to capture RGB and depth images of the scene. For this data set the object 6D pose was obtained with ArUco markers (Figure 5.3 shows an example of the ArUco markers used in this data set). An ArUco marker is a synthetic square marker composed by a wide black border and an inner binary matrix which determines its identifier. The

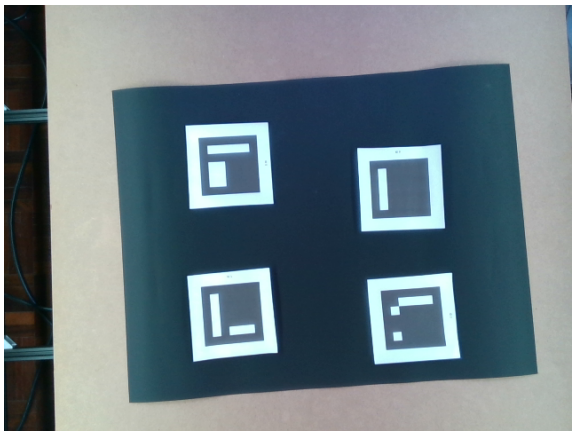


Figure 5.3: Example of Four ArUco markers that where used to identify object 6D poses.

black border facilitates its fast detection in the image and the binary codification allows its identification. The marker size determines the size of the internal matrix. For instance a marker size of 4x4 is composed by 16 bits. ArUco markers are bundle into dictionaries. The main properties of an ArUco dictionary are the dictionary size and the marker size.

The dictionary size is the number of markers that compose the dictionary. The marker size is the size of those markers (the number of bits that can be represented). The ArUco module present in OpenCV includes some predefined dictionaries covering a range of different dictionary sizes and marker sizes. One may think that the marker id is the number obtained from converting the binary codification to a decimal base number. However, this is not possible since for high marker sizes the number of bits is too high and managing such huge numbers is not practical. Instead, a marker id is simply the marker index within the dictionary it belongs to.

Within the module of the ArUco markers multiple methods and functions are available to detect them, obtain its identification, and with camera calibration and distortion calibrated it is possible to obtain the 6D pose of the ArUco markers thus generating the 6D pose ground truth for our data set.

Besides having the RGB images, Depth images and 6D pose annotations, the semantic segmentation were manually annotated using the Computer Vision Annotation Tool (CVAT). With the semantic segmentation annotated it was also possible to obtain bounding box annotations for the data set.

The data set is composed by 5100 scenes, this meaning that it has 5100 RGB and Depth images, 6D pose and Semantic Segmentation annotations for each object present in the scenes.

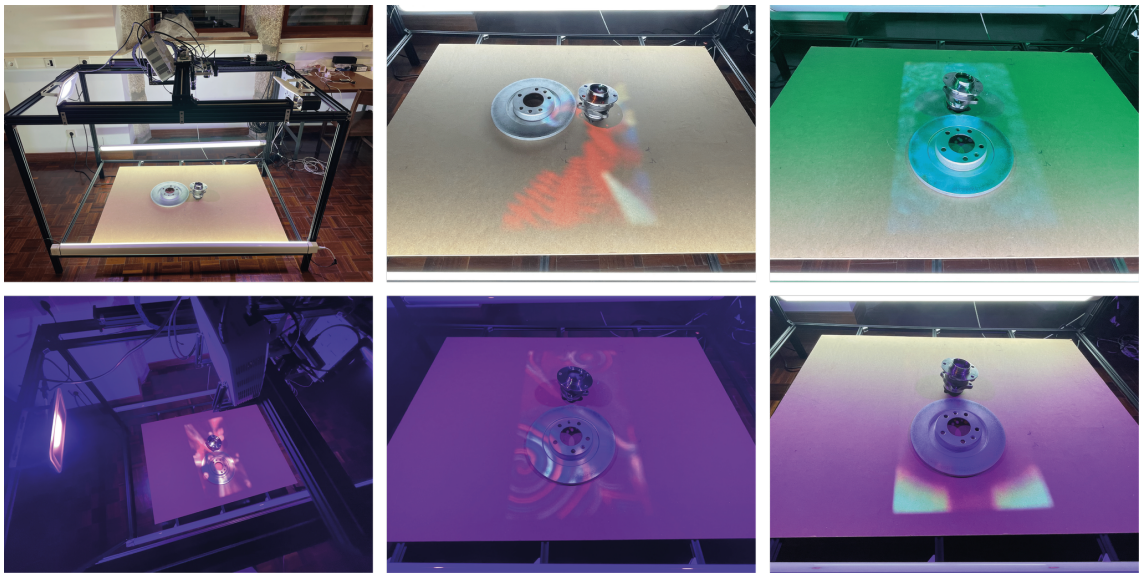


Figure 5.4: Pictures of the scanner system used to capture data sets with controlled light conditions/environment.

To create a robust data set that can simulate multiple real-world problems a system to capture the data was created (we named it scanner system, Figure 5.4 shows the scanner system). This system has multiple light sources, some of them can change color, it has a mechanical system that can move the camera in a 2D planar space above the surface and a projector that is also used to project patterns and can be another light source for the scene.

This system enabled the environment to be changed in terms of light positions and ambiance colors. Since most of the objects that are present in the data set are metallic and the light projection over them can difficult their detection and also invalidate the laser pattern that the RGB-D cameras used thus creating points of failure in depth distance readings. Also these objects can change its colors when the light source color changes, this initially might not look as big problem but is specially in the setting that the work we present is needed. For example, if there is a siren/hazard light in the factory signaling something nearby these objects they can change from gray to red or orange thus creating problems with the depth capture and with the algorithms used to detect and estimate their pose.

The data set is divided into three subsets, train, validation and test. There are 4000 scenes in the train subset, 1000 scenes in validation subset, and 100 scenes in test. For the train and validation subset the scenes were captured in the scanner system. For the test subset the scenes were captured in another different environment, where the camera is attached to a gripper of an Universal Robot 3 arm (Figure 5.7).

Table 5.1: Accuracies of our methods in our test subset and inference time per scene. Empty cells are metrics that cannot be shown due to the method being a full 6D pose estimation pipeline, thus not needing semantic segmentation sub tasks.

Method	Semantic Segmentation Pixel Accuracy (%)	6D Pose Estimation ADD-S (%)	Semantic Segmentation Time (s)	6D Pose Estimation Time (s)	Total Inference Time (s)
MaskedFusion	89.5	92.2	0.2	0.02	0.22
MPF6D	-	93.4	-	-	0.17

The test set is similar to the end usage of this system and completely different from the training and validation data. With this approach is possible to have a better idea of the performance in the real-world and since the test set is never shown to the neural network during the training phase it is a good subset to retrieve the performance metrics.

As shown in Table 5.1 our MaskedFusion pipeline achieved 89.5% pixel accuracy in the semantic segmentation task, and 95% accuracy (add-s (2.17)) in the 6D pose estimation task. To measure the performance of our method in this data set we used the add-s (2.17) metric because all objects present in it are symmetric. In the MPF6D method we achieved 93.4% add-s (2.17). Overall as shown in the Table 5.1 both methods can be used in real time.

5.1.2 Robot Manipulation

The Universal Robot 3 arm can be controlled using three different methods. The method that is commonly use to program and control it is named PolyScope. In PolyScope, is possible to control the robot in real-time or create routines and programs that can be later executed in a loop to execute the programmed functions, for example, object palletization. This method is the main supported method by Universal Robots, and this software comes installed in every robot. Besides this method, there are two more methods that can use an external computer to control the robot, Robot Operative System (ROS) and Real-Time Data Exchange (RTDE) communication. In ROS, multiple community members and the Universal Robot company developers have been developing drivers to control these robots, but these drivers do not work perfectly. We found multiple trajectory problems in the paths that ROS calculated based on these drivers and inverse kinematics present in ROS. Problems like self collision in the predicted path and even multiple not found paths that we know that were possible because when using the PolyScope and inputting the same end effector coordinates that we want to achieve, the trajectories work perfectly. Finally, the RTDE socket communication with the Universal Robot was the best method that we experimented where we could fully control the robot with an external computer. To use this method we developed a Python library that enable the easy integration of this method with other Python programs.

6D Pose Estimation and Object Recognition

This library was developed to enable interaction between the deep learning and machine learning Python programs and the robot, whereas these programs usually retrieve camera data and process it in order to extract information from the environment, estimate or define robot executions and finally send the actions to the robotic arm in order to move it. In this library we implemented multiple functions for movement, retrieving robot sensor data, and gripper control. The library has functions to send radian angles for each joint, enabling the user to put the robot in specific poses, and functions to move the end effector and the gripper to specific Cartesian coordinates, with origin in the robot base. The functions that exist in this library responsible to retrieve sensor data, output robot temperature, robot pose, robot joint forces, robot end effector and gripper Cartesian coordinates. To control the RG2 gripper, we developed three functions, that are used to open and close the RG2 fingers and to control diameter that the fingers keep when open.

5.1.2.1 Robot to People Awareness

During the development of this Python library, a collaboration project was done where the library was used to control the robot in a pick-and-place static routine, where three machine learning methods were involved. The objective of this project was to create a pick-and-place task that our robot could execute. In this pick-and-place task, the robot should be able to determine if there were items to be picked and how many. The number of items to be picked in the line was important information because that would influence the speed that the robot operates, thus meaning more objects in the queue, the faster the robot would move. For these two objectives, the robot library needed to be used to create all the robot routines of picking and placing (static positions of the picking and placing position with the aim of simulation an object picking from one roller carpet to another). A machine learning method was used to detect the objects and count the number of waiting objects. This method interacted with the robot control library in order to control the robot's speed for the movements and also to pause the robot in case no objects were waiting to be picked.

The third objective of this project was to detect and identify people near the robot's area of action and also control the robot to be stopped or slowed down when people were in the vicinity of the robot. For this last task, two methods were needed. The first method would detect people, using an RGB-D camera. This method provides a cubic detection and localization of the people from the camera's viewpoint. After the detection, a head crop was made, and a method for facial detection and recognition was used to find the person's face and match it against a database of known personnel. These detections influence how the robot operates because two zones of action were defined around the robot: a warning zone (yellow) and a critical zone (red), which can be seen in Figure 5.5. If someone is known and allowed to be near robots and enters the warning zone, no changes are made to the normal functioning of the robot, but that event is registered. If that type of person enters the red zone, the robot's velocity is reduced, and the event representing the person's

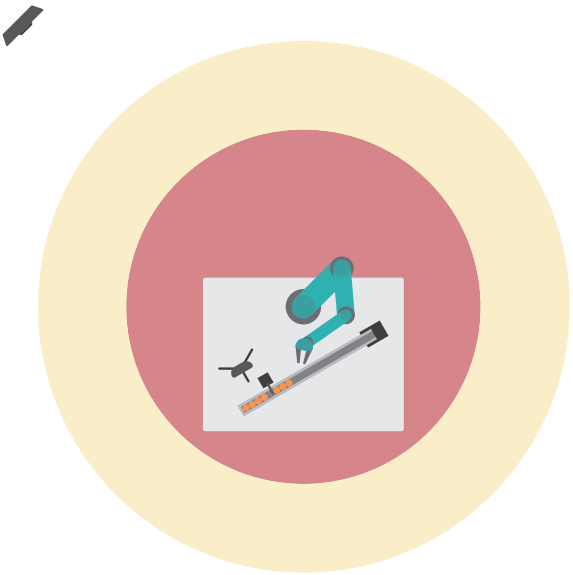


Figure 5.5: Robot workspace monitoring using a Kinect (top-left). Yellow represents the warning zone, red represents the critical zone.

detection and identification is registered. When the person is unknown or not allowed to be there and enters the warning zone, the robot's velocity is reduced, and if that person enters the critical zone, the robot stops, and all of the events are registered. The robot's speed-ups and speed-downs based on the object detection are also registered in the logs.

The collaboration in this project considered that these registered logs could be altered if some accident occurred. A blockchain was proposed to save all of these logs, avoiding modifications to the logs in these types of situations. This type of solution can be used in the real-world. Some companies may try to change logs to hide malfunctioning robots or software, thus avoiding responsibilities.

5.1.3 Calibration

To execute movements of the Universal Robot 3 robotic arm, a calibration between it and the RGB-D (Intel RealSense D415) camera that has the scenario viewpoint must be done.

To have a calibration between the two 3D spaces an translation vector and rotation matrix must be calculated. This will enable us to convert the camera Cartesian coordinates into the robot Cartesian coordinate space/world.

Table 5.2: Euclidean distance measurement of the error in millimeters between the camera point that we expect to reach and the robot achieved point.

# Sample Points	Average Error (mm)
4	19
14	12
24	10

To start the calibration process multiple pairs of coordinate points must be collected. This pair of points must correspond to the x,y,z coordinates from the camera and x,y,z of the robot world points. For our calibration, at least four pairs must be collected, but with more collected pairs a better precision of the calibration can be achieved.

To do some experiments in the calibration process we collected 24 pairs of coordinates points and then we randomly selected a set of four pairs of coordinates points, a set of 14 pairs and the set with all the 24 pairs. With these random sub-set of points we executed the calibration process and we measured the euclidean distance between the camera point that we wanted to reach and the real point that the robot reached in the real-world. The results of these experiments are present in the Table 5.2, where the first column shows the number of points present in the set and the second column shows the average euclidean distance in mm of the error between the expected point and the robot achieved point. Its possible to see that as the number of points increases, the average euclidean distance error decreases, as expected.

To find the translation vector and rotation matrix that will enable us to convert the camera points into the robot world, the Kabsch [Kab76] algorithm was used. The Kabsch algorithm is a method that calculates the optimal rotation matrix that minimizes the root mean squared deviation between two paired sets of points. The implementation of this algorithm that was used in our project is present on the Intel RealSense GitHub (`https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python/examples/box_dimensioner_multicam`).

After the Kabsch algorithm execution we have the translation vector and rotation matrix that will be multiplied by the camera points and then added to the translation vector to obtain the final Cartesian coordinates for the robot. With the result of these operations we have the robot coordinates that we will use to send the robotic arm to that point and execute a grasp. The coordinates that we want to convert to the robot coordinates are the coordinates that our 6D pose estimation method predicts. With the estimated object translation vector and rotation matrix converted to the robot world it is possible to move to the next step, object grasping.

Figure 5.6: Example of data set objects with the ideal grasping points represented in the pictures, X denotes the center of grasp and the 90º degrees rotated T represents the gripper fingers.

5.1.4 Object Grasping

Using the object 6D pose estimation obtained from the deep learning methods we can set the best grasping point in the 3D model to then estimate these best grasping points in the selected object of the scene. Depending on the object, the best grasping points will vary. For our data set we have two objects that can be picked up with the RG2 gripper (a two finger gripper) and two objects that cannot be picked up by our Universal Robot 3 arm due to its weight and size limitations. For these two last objects the picking instruction is to close the gripper and move it to center of the object. This idea of simulating this type of grasp comes from the gripper that will be in the factory to pick up these two objects (bearing and disk). The gripper that will be used in the factory is an electromagnetic gripper. The correct grasping for this type of gripper is placing them as near as possible to the object center point and then activate the electromagnet in order to pick the object. For the box and adapter we set up the best grasping point in the middle of the object where the longest part of the object fits in the middle of the RG2 fingers. Figure 5.6 shows examples of the best set grasping points for each object.

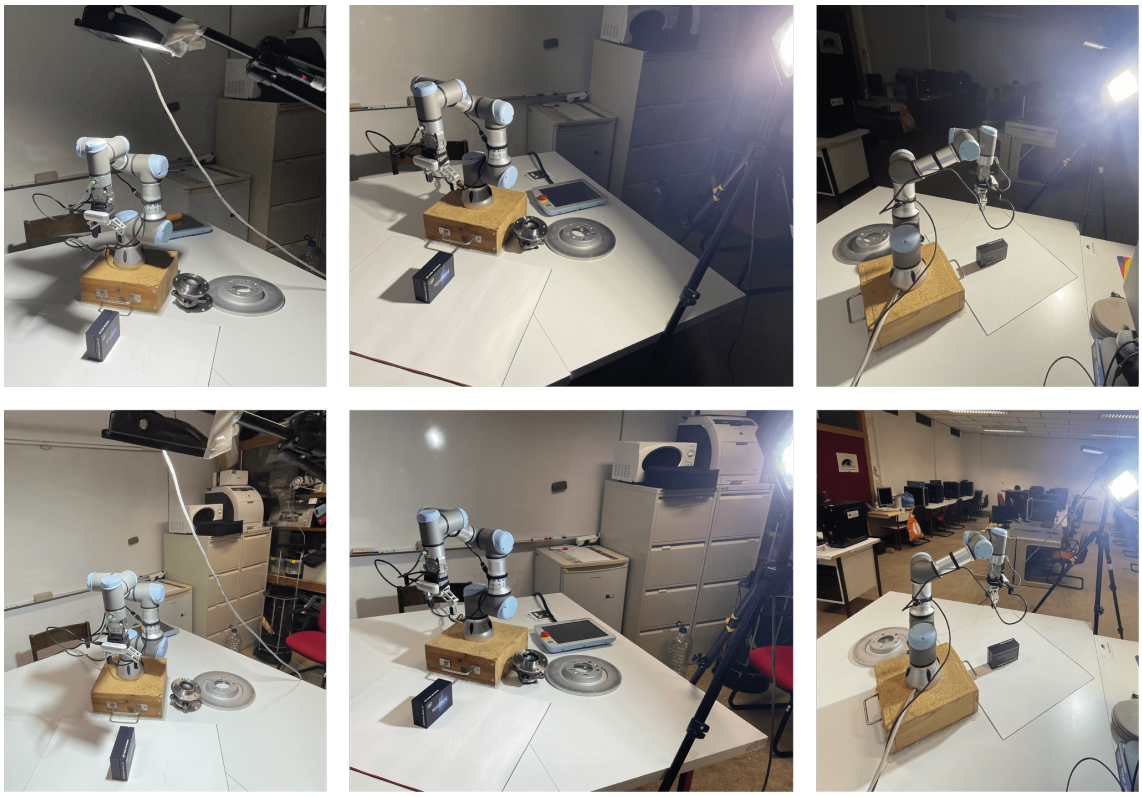


Figure 5.7: Pictures of some light conditions with the extra light source used in the real-world tests.

5.1.5 Real-world Results

To test our grasping pipeline an Intel RealSense D415 and the Universal Robot 3 were used to execute the pick-and-place task of the known objects presented in our data set. Just as a reminder, two of the objects could not be grasped due to their weight but we simulated its grasping by considering a good grasp when the robot gripper could reach the center of those objects. In our tests we define that a perfect grasp is when the robot can reach the grasping point within one centimeter of error. This error is mentioned here because it is the average error that we achieved in the calibration between the robot and the RGB-D camera, thus meaning even if all the methods estimated a good grasping point and this coordinate were sent to the robot it may have one centimeter of error due to the calibration. For the adapter and box object we also consider that these objects need to be picked in the estimated grasping point and without letting them fall.

For the tests we made 100 picking executions of the different scenes with one or more known objects, and 20 of the 100 scenes had occluded known objects. The video present in `https://youtu.be/M6nBkg3nmpI` shows some test runs. As shown in the Table 5.3 from the 100 test executions 10 executions failed. These fails where caused due to not finding the objects or the objects 6D pose. This mostly happened because of the light conditions, since in our tests we did not use any light source other than the normal ceiling lights and

Table 5.3: Real-world experiments with different light conditions.

Type of experiments	# of experiments	# pass experiments	# fail experiments	# of objects non detected	# of 6D pose estimations fail
non controlled environment	100	90	10	8	2
only ceiling lights	10	10	0	0	0
ceiling lights and one extra light source	20	10	10	8	2
only one light source	20	6	14	9	5

sun light available during the test hours. The other 90 test runs were considered correct picks, all of them withing one centimeter of error.

To confirm our theory of our fails coming from the light conditions we executed 50 extra tests with somewhat controlled light environment. This set of tests was executed during the night to avoid any solar light. From these 50 tests, 10 tests were executed with only the ceiling lights turned on, and all of these tests ended up with a correct grasping. Then 20 tests were executed using the ceiling lights and one extra white light source. Figure 5.7 shows examples of the light used in these tests. For these 20 executions with ceiling and extra light, 10 of the tests failed again with no object found or no 6D pose estimation found. For the last 20 executions we only used the extra light source without the ceiling lights. In this last experiment 14 of the 20 runs failed with 9 without finding objects and 5 without finding the objects 6D pose. Table 5.3 summarizes all executed tests.

5.2 An Efficient Pick-and-Place Pipeline based on Quantized Segmentation

In this section, a new approach for efficient pick-and-place based on quantized segmentation is presented. This approach improves upon the previews pipeline proposed in Section 5.1.

To solve this type of task, a combination of multiple deep learning methods needs to be used in a pipeline. The pipelines that solve this task usually have the following sub-tasks: capturing the data, object detection or image segmentation, object 6D pose estimation, object grasping detection, robot planing, and motion execution. Our new method speeds up the pick-and-place task by removing sub-tasks from the original pipeline and using quantization methods on the required deep learning methods. The proposed pipeline has four sub-tasks: capturing the data, image segmentation, object grasping detection, robot planing and execution. One of the advantages of our method is that only one of these

sub-tasks uses deep learning (image segmentation) and the deep neural network used is quantized in order to execute as fast as possible and enabling the method's deployment on edge computing devices. The previous pipelines used to execute unconstrained pick-and-place tasks require high-end GPUs during the inference to execute the predictions. In our pipeline the objective was to enable unconstrained pick-and-place without requiring high-end computers or GPUs.

5.2.1 Quantization

Neural network quantization reduces the precision of the used weights, biases, and activations such that they consume less memory. In other words, quantization reduces the size of a neural network's parameter representation, from typically 32-bit floats, to smaller, 8-bit integers. One clear advantage of quantization is a considerable decrease in memory use. For instance, going from 32-bit to 8-bit would result in a model size reduction of a factor of 4.

Quantization also has the potential to reduce network latency and improve power efficiency. Because operations can be carried out using integers rather than floating-point data types, network speed is increased. Most CPU cores, including those in microcontrollers, need fewer cycles to perform these integer operations that what they would require to make similar floating-point operations. Since the processing and memory access budgets are reduced, the overall power efficiency is enhanced.

Despite these advantages, the trade-off of quantization is that since neural networks are not representing information as accurately, their accuracy may suffer. However, it has been shown that quantization can frequently result in a minimal loss of accuracy [GKD+21].

5.2.1.1 Types of Quantization

In practice, there are two main approaches to quantization: post-training quantization and quantization-aware training. Post-training quantization, as the name suggests, is a technique in which the neural network is trained using floating-point computing and then quantized. To be quantized, the neural network is frozen after training, preventing further updates to its parameters, and those parameters are then converted, usually, to 8-bit integers.

This strategy, despite being straightforward, may result in an increased accuracy loss because all quantization-related errors happen after training and cannot be compensated.

Table 5.4: Accuracy results obtained to compare different methods for semantic segmentation. Bold values represent the highest accuracy.

Model (Backbone)	mIoU (%)	Pixel Acc (%)
SegNet	63.61	89.81
FCN (ResNet101)	64.80	90.23
U-Net	70.94	92.57
DeepLabV3	**82.33**	**98.25**

Quantization-aware training (QaT) [GKD+21] works to compensate for the quantization-related errors. It uses 32-bit floats during training, but the actual values used are a representation of 8-bit integers as 32-bit floats. By training the neural network using this representation in the forward pass during training, the quantization-related errors will accumulate in the total loss of the model during training, and the training optimizer will work to adjust parameters accordingly and reduce the error.

Quantization-aware training has the same benefits as post-training quantization but usually has a much lower accuracy loss than post-training quantization. The main disadvantage of quantization-aware training is that the model needs to be trained with this method using the CPU, thus requiring more time to train.

5.2.2 Semantic Segmentation Model

We compared DeepLabV3 [CPSA17] with U-Net [RFB15], SegNet [BKC17] and FCN (ResNet101) [LSD15] in our data set to choose the method that could achieve the best performance in it, to further apply multiple quantization methods and use it in the real-world. Table 5.4 shows the results of the comparison that we have made with our data set. We trained each model for 100 epochs and then the weights that performed the best in the validation sub-set were saved and loaded to execute in the test sub-set of our data set to obtain the table results. We can see that DeepLabV3 was the best performing method in the data set achieving the highest mIoU 82.33% and 98.25% of pixel-wise accuracy.

5.2.3 Proposed Method

A new pipeline was developed to solve unconstrained pick-and-place tasks. This pipeline consists of four sub-tasks where just one of the sub-tasks uses a deep learning method. The four sub-tasks are: capturing the data, image segmentation, object grasping detection, robot planing and execution. The sub-task that was solved with deep learning was the image segmentation task. To improve even further the speed of this sub-task, quantization methods were applied.

We use the DeepLabV3 model to execute the image segmentation sub-task because it achieved the highest accuracy in the comparison tests that were made in our data set that tries to represent as closely as possible our pick-and-place task.

Besides using the DeepLabV3 with 32-bit float representation we also tested the use of 8-bit integer representation of the parameters of the DeepLabV3 model. We have used both post-training quantization and quantization-aware training available in the PyTorch library. For the quantization-aware training we implemented a new hybrid approach. PyTorch quantization does not support GPU's and the model takes a long time to train in CPU. To speed up the training process we developed an hybrid method that can train the neural network mainly in GPU and then fine-tune the model parameters with quantization-aware training, thus enabling us to have the advantages of the quantization-aware training while not taking much more time to train the model for this quantization method. To test the Hybrid-QaT we setup three tests. One that relies on training 99% of the epochs using GPU and only 1% using QaT. Second test trains 95% of the epochs with GPU and 5% with QaT. Finally, 90% of the epochs train in GPU and 10% using QaT.

For the object grasping detection sub-task we start by finding the contours of the segmented object (object mask). The object mask is obtained with the DeepLabV3 semantic segmentation method. Then we use central *moments* to find the centroid of the object and its orientation.

Central *moments* are mathematical quantities used to describe various properties of an object in an image, such as its size, orientation, and shape.

The first-order central *moments* $mu10$ and $mu01$ are used to calculate the centroid of an image, which represents the center of mass or balance point of the object. They represent the distribution of pixel intensities along the x-axis and y-axis, respectively, and they are weighted by the pixel intensity values. By dividing these *moments* by the total pixel intensity ($m00$), we obtain the centroid coordinates (cx, cy) of the image.

The centroid (cx, cy) of the object is calculated as follows:

$$cx = \frac{mu10}{m00}, cy = \frac{mu01}{m00} \quad (5.1)$$

where $m00$ is the zero-order moment, which is simply the sum of all pixel intensities in the image.

Other three important central *moments* are $mu11$, $mu20$, and $mu02$, which represent the second-order central *moments* in the xy plane, along the x-axis, and along the y-axis, respectively.

To calculate the orientation of an object in an image we use the following equation:

$$\theta = \frac{1}{2} \times \arctan\left(\frac{2 \times mu11}{mu20 - mu02}\right). \quad (5.2)$$

The orientation theta is calculated based on the central **moments** $mu11$, $mu20$, and $mu02$. $mu11$ measures the distribution of intensity around the object's centroid and describes the object's tilt in the plane. $mu20$ and $mu02$ measure the spread of the object along the x-axis and y-axis, respectively, and describe the object's elongation in those directions.

With the central *moments* that enabled us to find the cx, cy of the object projected into a 2D plane and the θ angle we can find the last needed coordinate cz by the depth data in the cx, cy pixel, thus achieving all the needed coordinates and the object rotation. With this approach we can have a robust grasping method without using any deep learning technique.

5.2.4 Experiments

5.2.4.1 Semantic Segmentation with Quantization

The computer specs used for these experiments are: AMD Ryzen 5 3600, NVIDIA GeForce GTX 1080 Ti, 32GB of RAM and NVME SSD.

Table 5.5 shows all the results obtained for the DeepLabV3 method. We used the same settings for all four data sets.

When using Cityscapes data set most authors only use the 19 classes that have more instances in the data set but in this work we use all 30 classes since our main objective is not to compare with other semantic segmentation works but to evaluate different quantization methods.

We trained the method for 100 epochs for each experiment and we retrieved the time that it took to train (Training Time). Then during inference time, in the test subset of

Table 5.5: Experiment results for 100 training epochs. (*) represent estimation values that were calculated with the time that one epoch needed, multiplied by the 100 epochs required to train the method. Bold values are the best accuracy values per data set.

	Pixel Acc (%)	mIoU (%)	Inference Time (s)	Training Time	Size (mb)
Data set			Our data set		
GPU	98.25	82.33	0.01	05h39	225
CPU	98.25	82.33	2.14	3 days 13h43	225
Static	98.00	82.15	1.15	05h39	58
QaT	98.20	82.24	1.14	4 days 20h40	58
Hybrid-QaT 1%	**98.81**	**85.59**	1.14	06h46	58
Hybrid-QaT 5%	98.21	81.06	1.14	11h12	58
Hybrid-QaT 10%	98.78	85.41	1.14	16h45	58
Data set			LineMOD		
GPU	97.19	**85.67**	0.01	24h00	225
CPU	97.19	**85.67**	2.09	15 days 04h05	225
Static	96.87	84.99	1.16	24h00	58
QaT	96.98	85.28	1.14	20 days 15h33	58
Hybrid-QaT 1%	**97.24**	85.53	1.14	1 day 04h43	58
Hybrid-QaT 5%	97.18	85.53	1.14	1 day 23h35	58
Hybrid-QaT 10%	97.20	85.60	1.14	2 days 23h10	58
Data set			CityScapes		
GPU	88.57	**41.94**	0.01	11h50	225
CPU	88.57	**41.94**	1.67	7 days 11h30	225
Static	88.54	41.71	1.12	11h50	58
QaT	88.56	41.83	1.11	10 days 04h20	58
Hybrid-QaT 1%	**89.03**	40.88	1.11	14h10	58
Hybrid-QaT 5%	88.99	41.20	1.11	23h28	58
Hybrid-QaT 10%	88.99	41.36	1.11	1 day 11h05	58

each data set, we measured the inference time per image (Inference Time) and we show the accuracy for two evaluation metrics mIoU and pixel-wise accuracy (Pixel Acc). In the last column of the table we show how much memory the model requires.

Based on the obtained results from our tests, using the Hybrid-QaT technique can improve in some cases the accuracy while using less resources and being fast during inference. Post-training quantization is an alternative when the accuracy is not critical, since there is no additional training cost to produce the quantized network.

When comparing QaT with Hybrid-QaT it is possible to see that Hybrid-QaT can achieve better results while using less time to train the neural network.

5.2.4.2 Real-World Object Grasping

We deployed the proposed pipeline in the real-world to execute the pick-and-place task. For the real-world experiments we used an Universal Robot 3 arm, an Intel RealSense

Table 5.6: Real-world results of the proposed pipeline in an unconstrained pick-and-place task for four objects.

	Avg. Inference Time (s) [STDev]	Size (mb)	Grasping Success (%)
CPU	9.93 [0.10]	225	96
Static	5.61 [0.05]	58	96
QaT and Hybrid-QaT	5.59 [0.09]	58	96

D415 and a small computer device with the following specs: Intel i5-4300U CPU, 8GB of RAM and a SATA SSD.

In the real-world experiments we measured the average inference time of the method when running on CPU, when using post-training quantization and quantization-aware training. This average was collected during 50 executions of the pipeline, these executions had the object to grasp or simulate the grasping of the objects present in the data set that we created. For these 50 executions we counted how many resulted in successful grasps, and in Table 5.6 we show the obtained results.

In Table 5.6, it is possible to see that in the real-world experiments the difference in accuracy's obtained when using quantization methods was not relevant since the robot could achieve the same object grasping success rate. We can conclude that using quantization speeds up the pipeline while not loosing grasping rate.

5.3 Conclusion

This chapter was shown that leveraging a good and robust 6D pose estimation method enables a high pick-and-place accuracy. The proposed pipeline: capturing the data, object detection or image segmentation, object 6D pose estimation, object grasping detection, robot planing, and motion execution that our methods used, achieved a 90% of success picking in our non controlled environment tests. Leveraging our capture system to create different light conditions in our data set helped in the high accuracy in real-world with different light conditions, and we also shown that the data set should have more light conditions since when testing with other types of light conditions it was possible to see some flaws in the methods trained with the data set. When using these type of methods in real-world scenarios, a large data set with multiple light conditions and environment backgrounds is needed to achieve the best possible results.

In a collaboration work, an accurate 3D people detection and facial recognition was developed that leverages block chains to store logs of multiple robots in a factory. This method enables factories to remove the robot cages in order to have an open space but still

secure to humans. This system also has the advantage of just slowing down production instead of stopping it when authorized staff are in the robot workplace.

With the second proposed approach, we achieved an accurate object grasp without estimating the object's 6D pose. This method is faster and requires less computational power to execute an efficient pick-and-place task in the real-world with good performance.

The method achieved 96% accuracy while just using 5.59 seconds on average to output the grasping coordinates for one known object, in a computer with low computational power.

Chapter 6

Conclusion and Further Research

Throughout this thesis, several contributions were made to enable object 6D pose estimation and the unconstrained pick-and-place to be executed with high accuracy. This chapter presents the main conclusions that resulted from the research work performed. Furthermore, it provides directions for further research.

6.1 Conclusion

The major goal of this thesis was the development of object 6D pose estimation methods capable of improving the robotics unconstrained pick-and-place tasks. As described along this document, the successful development of such methods and pipelines requires the knowledge of multiple distinct research. Our focus was on the 6D pose estimation but to accomplish this task it was required to implement different methods for tasks like object detection and localization, semantic segmentation, robot planning and manipulation, and others. Nevertheless, we believe that the contributions presented represent a step-forward towards a better experience in tasks that require object 6D pose, like augmented reality and multiple robotic tasks.

This thesis presents a series of novel deep learning methods for robust 6D pose estimation in real-world scenarios. The MaskedFusion method showcased impressive results, achieving pose estimation accuracy of less than 6mm in the LineMOD dataset and demonstrating an AUC score of 93.3% in the challenging YCB-Video dataset. Although it required longer training time, MaskedFusion exhibited low inference time, making it a viable solution for real-time applications. It was also demonstrated that the utilization of different color spaces can enhance the accuracy of 6D pose estimation methods, alongside the use of better segmentation algorithms.

Furthermore, the proposed single feed-forward network (MPF6D) method emerged as the top performer, surpassing other approaches in accuracy and inference time. With an astounding 99.7% accuracy in the LineMOD dataset and 98.06% accuracy in the YCB-Video dataset, this method showcased its potential for high-precision 6D pose estimation.

Additionally, we developed object grasping methods that achieved a remarkable accuracy in pick-and-place tasks. The first proposed pipeline, including data capture, object detection, 6D pose estimation, grasping detection, robot planning, and motion execution, achieved a 90% success rate in non-controlled environment tests. Leveraging a diverse dataset with varied light conditions proved crucial for accurate performance in real-world scenarios. On a second approach we demonstrated an accurate object grasping without using 6D pose estimation, offering a faster execution and requiring less computational power. This method achieved a remarkable 96% accuracy with an average execution time of 5.59 seconds while using a laptop without NVIDIA GPU, showcasing its efficiency and practicality.

The work developed during this thesis achieved excellent performance in the most used data sets for this task and also in the real-world. The developed work can be used in the real-world, in fact, it is being used on the INDTECH 4.0 project (POCI-01-0247-FEDER-026653). The INDTECH 4.0 project had a research line related to unconstrained pick-and-place that we collaborated with and the goal was to allow a robot to pick different objects present in car factories and place them in production lines or even assembly them on other components, all of these while being aware of the people around the robot workplace.

Depending on the main objective we have two solutions, the first solution presented on section 5.1 enables high accuracy in a pick-and-place task because of the knowledge of the 6D pose of the object, but it requires a powerful computer with a high-end GPU to execute such task. The second solution (section 5.2), might have less accuracy but it requires way less computational power to execute and it is easier to use as well, since it requires less data and easier annotations in order to perform well in the real-world. In most cases, dealing with data constraints and the use of metallic objects, the second solution proposed (section 5.2) might be the best solution to use in the real-world.

6.2 Further Research

We think that further research in the topic of unconstrained pick-and-place tasks should avoid high computational pipelines, to allow the use of less powerful computational units that require less power and cooling, thus leading to more robust hardware systems and also enabling the use of these types of pipelines in smaller robots, for example household robots.

An accurate object 6D pose estimation will be important in the next years to enable better experiences in augmented reality and progress in the interpretation of the real-

world.

There are several key areas that researchers should be focusing on for future advancements.

Robustness and generalization are crucial aspects that need to be addressed. While current methods perform well in controlled environments, they often struggle when faced with real-world scenarios characterized by diverse lighting conditions, occlusions, and object variations.

Real-time and low-latency pose estimation is essential for real-world applications requiring immediate and responsive interaction. However, current pick-and-place methods often face challenges in achieving high accuracy while maintaining real-time performance. Future work should focus on reducing computational complexity or designing better architectures to enable real-time performance while improving accuracy.

Data efficiency is another area of interest in 6D pose estimation. Training deep learning models for pose estimation often relies on large annotated datasets, which can be time-consuming and costly to create. Future research needs to improve data efficiency, using techniques such as few-shot learning, meta-learning, or active learning. By leveraging these techniques, models can learn to estimate object poses accurately with limited annotated data, reducing the need for extensive datasets and probably obtaining better real-world performance.

Finally, the next step to optimize our latest pipeline would be to extract the silhouette/shape of an object without needing previous knowledge of the object, which is currently necessary in the form of training masks.

These proposed advancements will pave the way for more accurate and practical applications in robotics.

Bibliography

[Ale12] Luís A. Alexandre. 3D descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October 2012. 11, 12

[AVB+11] Aitor Aldoma, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu, and Gary Bradski. Cad-model recognition and 6Dof pose estimation using 3D cues. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 585–592. IEEE, 2011. 40

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. xxii, 33, 34, 66, 94

[BKM+14] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. In *European conference on computer vision*, pages 536–551. Springer, 2014. 32

[BM92] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. SPIE, 1992. 40

[BMK+16] Eric Brachmann, Frank Michel, Alexander Krull, Michael Ying Yang, Stefan 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*, pages 3364–3372, 2016. 32, 38

[CMW+17] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017. 31

[Cor] Intel Corporation. Intel realsense web page. `https://www.intelrealsense.com/`. Accessed: 2023-01-18. xxi, 8, 9

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 26

[CPSA17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 34, 66, 71, 94

[CSW+15] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The YCB object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517. IEEE, 2015. 27

[CTT+12] Changhyun Choi, Yuichi Taguchi, Oncel Tuzel, Ming-Yu Liu, and Srikumar Ramalingam. Voting-based pose estimation for robotic assembly using a 3D sensor. In *2012 IEEE International Conference on Robotics and Automation*, pages 1724–1731. IEEE, 2012. 46

[DUNI10] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *2010 IEEE Conference on Computer Vision and Pattern Recognition*, pages 998–1005. IEEE, 2010. 45

[FaLL17] Max Ferguson, Ronay ak, Yung-Tsun Lee, and Kincho Law. Automatic localization of casting defects with convolutional neural networks. pages 1726–1735, 12 2017. xxi, 20

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. xxi, 12, 17, 18

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 31

[GDDM16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):142–158, 2016. 35

[GIRL03] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, and Marc Levoy. Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 260–267. IEEE, 2003. 41

[GKD+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2021. 93, 94

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. 31

[HCI+11] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE transactions on pattern analysis and machine intelligence*, 34(5):876–888, 2011. 32, 46

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017. xxii, 36, 37

[HHC+11a] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 international conference on computer vision*, pages 858–865. IEEE, 2011. 26, 27, 47, 52, 59

[HHC+11b] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 international conference on computer vision*, pages 858–865. IEEE, 2011. 32

[HHRF11] Evan Herbst, Peter Henry, Xiaofeng Ren, and Dieter Fox. Toward object discovery and modeling via 3-d scene comparison. In *2011 IEEE International Conference on Robotics and Automation*, pages 2623–2629. IEEE, 2011. 40

[HLI+12] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012. 46

[HSH+20] Yisheng He, Wei Sun, Haibin Huang, Jianran Liu, Haoqiang Fan, and Jian Sun. PVN3D: A deep point-wise 3D keypoints voting network for 6Dof pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. xxii, 45, 47

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 20, 50, 54

[Kab76] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, Sep 1976. Available from: `https://doi.org/10.1107/S0567739476001873`. 82, 89

[Kat03] Dimitrios Katsoulas. Robust extraction of vertices in range images by constraining the hough transform. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 360–369. Springer, 2003. 32

[KBM+15] Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In *Proceedings of the IEEE international conference on computer vision*, pages 954–962, 2015. 32, 46

[KHG+19] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9404–9413, 2019. 25

[KMT+16] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep learning of local RGB-D patches for 3D object detection and 6D pose estimation. In *European Conference on Computer Vision*, pages 205–220. Springer, 2016. 43

[KMT+17] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1521–1529, 2017. 27, 47, 61

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 33

[LAP22] Vasco Lopes, Luís A. Alexandre, and Nuno Pereira. Controlling robots using image analysis and a consortium blockchain. In *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 637–642, 2022. 4

[LBRF11] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A scalable tree-based approach for joint object and pose recognition. In *Twenty-fifth AAAI Conference on Artificial Intelligence*, 2011. 40

[LDG+17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017. xxii, 21, 22

[LFPA21] V. Lopes, M. Fernandes, N. Pereira, and L.A. Alexandre. A time-segmented consortium blockchain for robotic event registration. In *The 3rd International Conference on Blockchain Technology (ICBCT 2021)*, Shanghai, China, 2021. 4

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014. 66

[LPA19] Vasco Lopes, Nuno Pereira, and Luís A. Alexandre. Robot workspace monitoring using a blockchain-based 3D vision approach. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2812–2820, 2019. 3, 81

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 3431–3440, 2015. xxii, 36, 37, 41, 66, 94

[LWJ+18] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6D pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698, 2018. 47

[MAFK17] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3D bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017. 32

[MS15] Daniel Maturana and Sebastian Scherer. Voxnet: A 3D convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. 32

[MUS15] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for augmented reality: a hands-on survey. *IEEE Transactions on Visualization and Computer Graphics*, 22(12):2633–2651, 2015. 31

[NIH+11] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. IEEE, 2011. 40

[Nis21] Takato Nishijima. Universal approximation theorem for neural networks. *arXiv preprint arXiv:2102.10993*, 2021. 18

[ORL18] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making deep heatmaps robust to partial occlusions for 3D object pose estimation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 125–141, Cham, 2018. Springer International Publishing. 39

[PA19] Nuno Pereira and Luís A. Alexandre. Using pcl gobal descriptors in a dense-fusion architecture. In *25th Portuguese Conference on Pattern Recognition (RECPAD)*, 2019. 4, 49

[PA20a] Nuno Pereira and Luís A. Alexandre. Exploring the impact of color space in 6D object pose estimation. In *26th Portuguese Conference on Pattern Recognition (RECPAD)*, 2020. 4, 49

[PA20b] Nuno Pereira and Luís A. Alexandre. MaskedFusion: Mask-based 6D object pose estimation. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 71–78, 2020. 4, 26, 49, 65, 68, 69, 70, 71, 82

[PA21] N. Pereira and Luís A. Alexandre. Impact of segmentation and color spaces in 6D pose estimation. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, Santa Maria da Feira, Portugal, 2021. 4, 49

[PA22] Nuno Pereira and Luís A. Alexandre. An efficient pick-and-place pipeline based on quantized segmentation. In *International Conference on Machine Learning, Control, and Robotics (MLCR 2022)*, October 2022. 4, 81

[PA23] Nuno Pereira and Luís A. Alexandre. MPF6D: Masked Pyramid Fusion 6D Pose Estimation. *Pattern Analysis and Applications*, May 2023. Available from: `https://doi.org/10.1007/s10044-023-01165-9`. 4, 26, 49, 82

[PGBP10] In Kyu Park, Marcel Germann, Michael D Breitenstein, and Hanspeter Pfister. Fast and automatic object pose estimation for range images on the GPU. *Machine Vision and Applications*, 21(5):749–766, 2010. 46

[PHP+12] Chavdar Papazov, Sami Haddadin, Sven Parusel, Kai Krieger, and Darius Burschka. Rigid 3D geometry matching for grasping of known objects in cluttered scenes. *The International Journal of Robotics Research*, 31(4):538–553, 2012. 46

[PLH+19] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. PVNet: Pixel-wise voting network for 6DoF pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4561–4570, 2019. 27

[PZC+17] Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G Derpanis, and Kostas Daniilidis. 6-DoF object pose from semantic keypoints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2011–2018. IEEE, 2017. 32

[QLW+18] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3D object detection from RGB-D data. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pages 918–927, 2018. 32

[QSMG17a] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. xxii, 43, 45, 50, 53, 54, 58

[QSMG17b] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 32

[RBTH10] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010. 11, 40, 43, 52, 53, 54

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011. 41

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 38

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. xxii, 33, 94

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. Available from: `https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf`. 31, 36, 37, 44

[RL17] Mahdi Rad and Vincent Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *Proceedings of the IEEE international conference on computer vision*, pages 3828–3836, 2017. 32, 39, 47

[SMD+18] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3D orientation learning for 6D object detection from RGB images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018. 32, 47

[SRKB11] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *2011 IEEE International Conference on Robotics and Automation*, pages 2601–2608. IEEE, 2011. 43

[SYJ18] Zhiqiang Sui, Zhefan Ye, and Odest Chadwicke Jenkins. Never mind the bounding boxes, here's the sand filters. *arXiv preprint arXiv:1808.04969*, 2018. 44

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 39, 41

[TAJF19] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5229–5238, 2019. 34

[TSF18] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6D object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301, 2018. 32

[TTKK14] Alykhan Tejani, Danhang Tang, Rigas Kouskouridas, and Tae-Kyun Kim. Latent-class hough forests for 3D object detection and pose estimation. In *European Conference on Computer Vision*, pages 462–477. Springer, 2014. 32

[VLM18] Joel Vidal, Chyi-Yeu Lin, and Robert Martí. 6D pose estimation using an improved method based on point pair features. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 405–409. IEEE, 2018. 26

[WV11] W. Wohlkinger and M. Vincze. Ensemble of shape functions for 3D object classification. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 2987–2992, Dec 2011. 11, 52, 53, 54

[WXZ+19] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6D object pose estimation by iterative dense fusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3343–3352, 2019. xxii, xxv, 26, 27, 29, 44, 45, 47, 49, 50, 58, 59, 61, 62, 63, 68, 69, 70

[WZH+19] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. FastFCN: Rethinking dilated convolution in the backbone for semantic segmentation. *arXiv preprint arXiv:1903.11816*, 2019. 33

[XAJ18] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3D bounding box estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2018. 47, 50, 58, 61

[XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017. xxii, 26, 27, 29, 41, 42, 47, 50, 59

[YWC05] Ying Kin Yu, Kin Hong Wong, and Michael Ming Yuen Chang. Pose estimation for augmented reality applications using genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(6):1295–1301, 2005. 31

[ZDY+14] Menglong Zhu, Konstantinos G Derpanis, Yinfei Yang, Samarth Brahmbhatt, Mabel Zhang, Cody Phillips, Matthieu Lecce, and Kostas Daniilidis. Single image 3D object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3936–3943. IEEE, 2014. 31

[ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. xxi, 12, 15, 18, 19, 21

[ZSQ+17] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2881–2890, 2017. 34, 72

[ZT18] Yin Zhou and Oncel 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*, pages 4490–4499, 2018. 32

[ZYS+17] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1386–1383. IEEE, 2017. 41