UNIVERSIDADE D
COIMBRA

Francisco Rodrigues Lourenço

6DoF Object Pose Estimation from RGB-D Images Using Machine Learning Approaches

Dissertação no âmbito do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores, na especialização de Automação, orientada pelo Professor Doutor Helder Araujo e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologias da Universidade de Coimbra.

Junho de 2021

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

6DoF Object Pose Estimation from RGB-D Images Using Machine Learning Approaches

Francisco Rodrigues Lourenço

June 2021

deer.uc
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
E DE COMPUTADORES

INSTITUTE OF SYSTEMS AND ROBOTICS
UNIVERSITY OF COIMBRA

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor, Prof. Helder Araujo. Considering the current pandemic and that this thesis was elaborated on during the quarantine, I could not be more grateful for the massive effort my supervisor put into giving me the best advice, being available at all times, and making all needed materials available. Thanks to him, quarantine was not a hurdle.

I also want to thank my parents for being patient, supportive, and fund me during my studies. Thanks to them, I am not making popcorn at the local cinema theatre anymore.

I am also grateful for the advice provided by the machine learning expert Eng. Mariana Lourenço, which was crucial in solving the more complex implementation-related problems I had during this thesis.

My thanks also go to my friends. For their companionship, life advice, availability, and for keeping me sane during the past years. They definitely made my life better.

I also have to thank my colleague Eva Curto, for providing me materials, advice, and continuity to the Intel Realsense Experimental Evaluation research.

Abstract

Object pose estimation using RGB-D images has gained increasing attention in the past decade with the emergence of consumer-level RGB-D sensors in the market. Their low-cost coupled with relevant technical specifications led to their application in areas such as autonomous driving, augmented reality, and robotics.

Depth information has, in general, brought additional complexity to most applications that previously used only RGB images. Moreover, when trying to estimate an object pose, one may face challenges such as cluttered scenes, occlusion, symmetric objects, texture-less objects, and low visibility due to insufficient illumination. Accordingly, researchers started to adopt machine learning approaches to tackle the 6DoF [1] of the object pose estimation problem. Such approaches are often quite complex to implement and computationally demanding. Furthermore, the research was only directed to RGB-D videos quite recently, with the first benchmark dataset containing videos being published only in 2017. Therefore, only very recent methods were designed to process videos, and some questions regarding real-time applicability arise.

That being said, this thesis aims to explore all the tools required to build a 6DoF pose estimator, provide a comprehensive review on each tool, compare and evaluate them, assess how a practitioner can implement such tools, evaluate whether or not it is possible to estimate 6DoF poses in real-time, and also evaluate how these tools generalize to a real-world scenario. As a plus, it will be proposed the usage of directional statistics to evaluate an RGB-D sensor precision, a tweak to a famous 6DoF object pose estimation model, a pipeline that uses a novel 3D point cloud registration algorithm to aid the pose estimator, and a metric that can measure the precision/repeatability of both estimated poses of a model and the ground-truth poses of a dataset.

Keywords: 6DoF Object Pose Estimation; 3D Point Cloud Registration; Computer Vision; Machine Learning; RGB-D Sensors;

[1] Degrees of Freedom

Resumo

A estimativa da pose de objetos em imagens RGB-D, tem ganho bastante atenção na passada década com o aparecimento de sensores RGB-D ao nível do consumidor. O seu baixo custo acoplado com relevantes especificações técnicas, levaram à sua aplicação em áreas cientificas tais como condução autónoma, realidade aumentada e robótica.

Em geral, a informação de profundidade trouxe complexidade adicional a grande parte das aplicações práticas onde se usavam apenas imagens RGB. Para além disso, quando se tenta estimar a pose de um objeto, há outros desafios tais como cenas com vários objetos, oclusão por parte dos mesmos, objetos simétricos, objetos sem textura e até falta de visibilidade devido a pouca iluminação. Tendo isto em conta, os investigadores começaram a adoptar técnicas de aprendizagem automática para resolver o problema da estimação da pose de objetos. O problema com esta abordagem é que, por norma, costuma ser computacionalmente intensiva e complexa de implementar. Para além disso, apenas recentemente a investigação se tem direcionado para vídeos RGB-D, com o primeiro dataset de referência contendo apenas vídeos a ser publicado em 2017. Portanto, apenas poucos e bastante recentes métodos foram desenvolvidos para funcionar com vídeos, tornando assim o funcionamento em tempo real numa questão ainda por resolver.

Posto isto, esta tese tem como objectivo explorar todas as ferramentas necessárias para construir um estimador da pose, oferecer uma revisão compreensiva para cada uma destas ferramentas, comparar e avalia-las, estudar como estas podem ser implementadas, avaliar se a estimação da pose poderá ser ou não feita em tempo real e também como esta se generaliza para o mundo real. Em adição a isto, será proposto o uso de estatística direcional para o avaliação da repetibilidade de sensores RGB-D, um melhoramento na estrutura de um bastante conhecido estimador da pose, uma arquitetura que utiliza um algoritmo de aproximação geométrica bastante recente como auxílio ao estimador da pose, e ainda uma métrica que permite avaliar a repetibilidade tanto das poses estimadas como das poses fundamentais de um dataset.

Palavras-chave: Estimação da pose de objetos; Aproximação Geométrica; Visão por Computador; Aprendizagem Automática; Sensores RGB-D;

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Due to its usefulness and high practical applicability, 6DoF [1] object pose estimation is a widely researched subject in robotics and computer vision. This problem consists of retrieving a rotation and a translation of the target object coordinate frame with relation to the camera coordinate frame, having only RGB, D, or RGB-D [2] images as its inputs. These images can be obtained by cameras that can or cannot be attached to the robot. Prior methods only used 2D [3] RGB images ([6],[7],[8],[9]) to solve the 6D pose estimation but, with the rise of RGB-D technology and growing accessibility of RGB-D sensors, recent methods such as ([10],[11],[12],[13]), are mainly using this technology. Depth information brought more room for research in this field, allowing these methods to be much more accurate and robust. However, more computational power is needed, requiring sophisticated techniques to process the data and turning real-time applicability into a new challenge.

As long as we know an object absolute pose in the camera coordinate frame, we can infer its pose in relation to whatever location we want, for instance, relative to an end-effector of a robot. To understand the importance of this task, we can think about a robot that tries to pick up an object from the floor. To make this possible, the robot not only needs to know where the object is (a three-dimensional translation), but also how it is rotated (a three-dimensional rotation) in relation to its end-effector. In addition to this example, 6DoF pose estimation has many other practical applications, such as autonomous driving, SLAM

[1] DoF - Degrees of Freedom

[2] "D" here, stands for Depth

[3] "D" here, stands for Dimensions

[4], AR [5], and VR [6].

Some of the main adversities that one may find in estimating an object's 6DoF pose are that in most practical applications, we might face cluttered scenes (scenes with several objects), occlusion (overlapping of the objects in the image frame), texture-less objects, symmetric objects and low visibility due to poor illumination. This means that the task of a pose estimation method is not only to estimate a 6DoF pose but also to do this for several objects while not having full visibility of them.

All these aspects have motivated researchers to start using machine learning to tackle the 6DoF object pose estimation problem. Therefore, many machine learning based approaches have been proposed in recent years, some through the usage of CNN's [7] [14], random forests [9], Hough forests [15] or even support vector machines [11]. In these methods, data consisting of images containing one or more objects instances and their respective poses is fed into a machine learning model for training, ending up with a well-trained system that can estimate an object 6DoF pose in real-time.

One of the main challenges of this research problem is, in fact, to obtain highly accurate pose estimation in real time.

Most of the research on 6DoF pose estimation using RGB-D images was performed in the last decade when the first consumer-level RGB-D sensors got into the market [16] and, even more recently, using RGB-D videos when the first benchmark dataset (YCB-Video [17]) was proposed, in 2017. Accordingly, most developed techniques to solve this problem were designed to work only with individual RGB-D images [9],[10],[11],[12]. Therefore, real-time applicability was not always the focus of recent research, where most of the proposed work relies upon computationally demanding post-processing algorithms to refine the estimated pose. As a result of that, we still do not know whether or not 6DoF pose estimation can be done in real-time or, in other words, if we can estimate an object pose in subsequent frames of a video stream.

If we now consider that a pose estimator cannot perform in real-time, it would also be interesting to study possible solutions to this problem. In this thesis, there is the belief that a 6DoF pose estimator's inference time can be improved with the help of 3D point cloud registration by performing pose tracking. In this approach, instead of solely using a pose estimator to compute absolute poses of the objects in all frames of a video, point cloud registration is used to track the object poses in subsequent frames, thus avoiding the usage of a possibly harder to compute algorithm in every frame. Hence, this topic will also be explored.

[4] Simultaneous Localization and Mapping
[5] Augmented Reality
[6] Virtual Reality
[7] Convolutional Neural Networks

In addition to the the real-time problem, there are some other aspects concerning 6DoF object pose estimation that this thesis aims to investigate. Considering the growing complexity of the pose estimation problem, machine learning models are getting large and complex due to the already mentioned adversities. Hence, it is important to understand how a practitioner can implement such models, use them for its own practical application, and evaluate how it generalizes to a real-world scenario. Moreover, it is also relevant to perform a comparative analysis on the tools needed to do so, namely, the RGB-D sensors.

1.2 Objectives

This thesis aims to answer the following questions:

- Which low cost RGB-D sensors are the best fit for the 6DoF object pose estimation, and how can we evaluate them?
- How straightforward is it to implement a machine learning based 6DoF pose estimator model? Can it perform in real-time?
- Is it possible to improve a 6DoF pose estimator inference time using 3D point cloud registration?
- How can we evaluate a 6DoF pose estimator precision/repeatability?
- How does a 6DoF pose estimator generalize to a real-world scenario? If not, how does it perform after training it using transfer learning?

To do so, all tools needed to estimate an object 6DoF pose will be covered, starting with an experimental evaluation of three recent RGB-D sensors. Then, a 6DoF pose estimator will be chosen, implemented, trained, and evaluated both in terms of accuracy and inference time. A 3D point cloud registration algorithm will be chosen and fused with the pose estimator model looking forward to evaluating whether inference time can be improved with this technique or not. Precision/repeatability will be evaluated by measuring the change in the relative poses of the objects in the scene. Finally, an experimental setup will be built, so the implemented models can be tested in a real-world scenario.

1.3 Thesis Structure

This thesis starts with Chapter 2, which describes the state of the art, covering 6DoF object pose estimation, RGB-D sensors, and 3D point cloud registration.

The following Chapters are presented as a compilation of four different studies performed on the 6DoF object pose estimation field, except for Chapter 4 where the implementations of the models used in Chapters 5, 6, and 7, are described. A tweak to a well-known 6DoF pose estimation method is also proposed.

The first study, in Chapter 3, was published at the VISIGRAPP 2021 conference and presents a comparative analysis of three RGB-D sensors from Intel, the RealSense SR305, D415, and L515. As aforementioned, Chapter 4 describes how the models and methods are implemented. In Chapter 5, a novel 3D point cloud registration algorithm is analyzed with the aim of fusing it with a 6DoF pose estimator, looking forward to building a fast pose tracker. Chapter 6 presents a study where the precision/repeatability of a 6DoF pose estimator is investigated, where a metric for 6DoF pose precision measurements is also proposed. Chapter 7 is a case study where all the work done comes together in order to evaluate how the implemented models generalize to a real-world scenario.

Finally, Chapter 8 draws the final remarks.

Chapter 2

State of the Art

2.1 6DoF Object Pose Estimation

As mentioned in Chapter 1, this topic has been heavily researched in the last decade due to the evolution of machine learning approaches and the availability of new technologies such as RGB-D cameras. Therefore, it is fair to say that this is still a relatively new topic in computer vision and robotics. Previous review-related studies addressed this problem at 2D level using RGB images, and because of that, there is much information on this field. On the other hand, as the authors of [18] claim, their work was the first comprehensive and most recent review of the methods on 6DoF object pose recovery. In this paper, the authors do a systematic literature review of all the work done in 6DoF pose estimation in the past decade, addressing RBG, D (using only depth information), and RGB-D based methods. Hence, the revision of the literature in this section uses their work as a basis. In the end, very recent methods that were published this year will also be mentioned as these are not considered in the mentioned literature review.

2.1.1 Estimation Levels

6DoF pose estimation can be done in two different levels:

- Instance-level - When a method is said to work at the instance-level, it means that, to estimate the 6DoF pose of an object, such a method will look specifically for a known object instance, i.e., an object used in the training phase.
- Category-level - As opposed to instance-level, category-level approaches deal with unseen objects.

Instead of precisely identifying the object they want to look for, these methods work with categories, e.g., cars, bicycles, boxes, toys. For example, while at the instance-level, the methods know precisely the brand, model, and color of a car whose pose they want to estimate, at category-level, the only information the methods have is that they should look for a car. Estimating 6DoF object pose at the category-level is usually a more complex problem to solve, but methods that work at this level generalize better than at the instance-level, as it might be possible to estimate the pose of a broader range of unseen objects instances.

2.1.2 6DoF Pose Estimation Categories

6DoF pose estimators can be divided into two main categories:

- 3D bounding box detectors - These methods work at the category-level and do not estimate the 6DoF pose directly but, instead, these fit a 3D bounding box to the object. To do so, these methods produce oriented 3D BBs [1] centered at some point $x = (x, y, z)$, size $d = (d_w, d_h, d_l)$ with orientation (θ_y). The bounding box can then be extended to the 6DoF space where the pose can be recovered.
- Full 6DoF pose estimators - Directly estimate the 3D translation and 3D rotation. Typically, these methods work at the instance-level. However, recent proposed full 6DoF pose estimators such as [19] address the category-level problem.

2.1.3 Mathematical Models

There are five different main approaches often followed to model the problem mathematically:

- Classification - Classification-based methods take a dataset of images of real or synthetic objects with the corresponding labels with the objects' ground-truth pose. On the testing phase, these methods hypothesize an object's pose by classifying it as one of the previously learned poses. In these, the 6DoF pose estimation is done as a discrete function of the input images.
- Regression - The training phase of these methods is similar to that of the classification-based methods but, in the testing phase, instead of using classifiers to estimate the pose, it uses regressors. Here, the 6DoF pose estimation is done as a continuous function of the input images.

[1] Bounding Boxes

- Classification and Regression - As the name suggests, this type of method uses both classification and regression for 6DoF pose recovery. For instance, classification can be used for a coarse pose estimation that can then be refined with regression or vice-versa. Classification and regression can also be done in a single shot, e.g., classification can be used to estimate the 3D translation and regression for the 3D rotation. Though the problem might be more complex than this, it serves as an illustrative example.

- Template Matching - The main idea of template matching modeling is the following: having a database of different views of some object (templates), sliding windows are run on top of the input image searching for an object instance. Each window is represented by feature descriptors. The windows are compared *a posteriori* with the templates and some metric is used to compute the distances (difference) between the windows and the templates. The 6DoF pose parameters of the template that scores the minimum distance from the windows is then assigned to the object instances and, by doing so, a rough estimate of the object 6DoF pose can be computed and then refined in further stages of the methods.

- Point-pair feature matching - This model is similar to template matching but, instead of using templates, feature points are extracted from training images containing an object of interest. One feature point is selected as a reference from the extracted points, and the remaining points are paired with this reference point. The resulting point-pair features are then stored in a database. On an online phase, point-pair features are extracted from the test image that will be later compared with the information on the database looking for matching features and, when a match is found, the relative pose of the object instance and the stored model can be computed, i.e., the translation and rotation transformations between matched point-pair features. If the 6DoF pose of the trained model is known, then the 6DoF pose of the object instance can be recovered.

2.1.4 Methods

The review-based paper [18] addresses RGB/D/RGB-D based methods, but only RGB-D based will be presented in this thesis. RGB-D is chosen over RGB or just depth information because, as the results obtained on [18] show, RGB-D based 6DoF object pose estimators outperform other methods, as RGB-D images gather more information from the objects than the other. As we have more information, we might be able to better describe the objects' features as well as their distance from the camera, especially when dealing with texture-less objects where RGB/D based methods underperform the most.

The following list describes shortly each considered method in [18] plus 7 methods that were not consid-

ered in the review-based paper but will be mentioned in this work due to their interesting performance claims.

✣ Methods in [18]:

- ✣ Brach et al. RGB-D variant 2016 [9] - Instance-level/Classification&Regression/Full pose estimator. This method is originally designed to estimate poses from RGB images but the authors also present in their work a variant where they adapt it by opening a depth channel and consequently estimating the 6DoF pose from RGB-D inputs. The method consists of three individual stages. At the first stage, object labels and object coordinates are jointly predicted by random forests, the uncertainty of the predictions are then reduced as much as possible by extending the random forest to an auto-context random forest with L_1 -loss regularization. In the second stage of the method, poses of multiple objects are estimated from the predicted 2D-3D correspondences. In the third stage the poses are finally refined by exploiting the uncertainty of object coordinate predictions.

- ✣ Kehl et al. [10] - -/Regression/Full pose estimator. This method does not work at instance-level nor category-level. It cannot be fit into these two categories due to its novel approach to the 6DoF pose estimation problem. Instead of dealing with objects as a whole, this method separates the object's appearance into its local parts and lets them vote independently for a 6DoF pose. To do so, in the detection phase, the method samples random patches from the input image that will be fed into an already trained convolutional autoencoder with the goal of descriptor regression. The outputted data from this network is then compared with data that was stored in a database on the training stage, where each entry holds a 6DoF pose vote. By matching the network outputs with the database information through a k-nearest neighbor search, each patch will vote for a 6DoF pose. A notable feature of this method is that it generalizes well to previously unseen data.

- ✣ Cabrera et al. [11] - Instance-level/Template matching/Full pose estimator. This method trains a SVM [2] to classify which images from a given set of negative images must be used as templates. The SVM is also used to further classify which parts of the templates are more relevant to latter compare them with test templates. The templates are used for object detection and, if a 6DoF pose is assigned to the templates, these can vote for an object's instance 6DoF pose.

- ✣ Linemod [12] - Instance-level/Template matching/Full pose estimator. This approach is quite simple and focuses on solving the problem of template matching-based methods where the templates are built online from the RGB-D outputs and require physical interaction of a human operator or a robot

[2] Support Vector Machine

with the environment. The templates are built automatically from CAD 3D models and later used to detect the objects and estimate their pose.

- Hodan et al. [13] - Instance-level/Template matching/Full pose estimator. This method aims to solve the problem associated with the excessive computational complexity of sliding window approaches. A solution to this problem is achieved by using a cascade-style evaluation of the window locations. At the first stage, fast filtering is performed, rejecting most of the locations. For each remaining location, candidate templates are selected through a voting procedure. The obtained templates are then verified by matching feature points. Once a successful match against a template is achieved, a rough estimate of the object's pose can be estimated. The roughly estimated pose is refined using a stochastic population-based optimization scheme known as Particle Swarm Optimization (PSO) on the final stage of the method.

- Hashmod [20] - Instance-level/Template matching/Full pose estimator. This method tries to solve the scalability problem of most template-based methods where just a few objects can be detected. This problem exists and is hard to solve because a huge amount of templates is needed to accurately estimate each object pose. To solve this problem, Hashmod proposes the use of hashing techniques to make the template matching process more efficient. Hence, the templates are stored in a hash table and, on the online phase, the image descriptors are subject to a hash function and thus retrieving correct templates very efficiently.

- Hinters et al. [21] - Instance-level/Point-pair feature matching/Full pose estimator. This method is based on [22] where point feature clouds are computed from the objects of interest and then pairs are formed. By doing so, it is possible to understand the relative pose between point pairs and, by comparing point clouds with previous data obtained on an offline stage, it is possible to retrieve the 6DoF object pose. According to [21], the problem with that method is how they sample the points to be paired, where it rejects points that are too close to each other, as these points tend to have similar normals and generate non-discriminative PPFs [3]. The authors of [21] state that we might be rejecting important information by sampling the points this way. Therefore, instead of sampling points by its relative distance, this method accepts points even if they are too close to each other as long as the angle formed by their normals is larger than 30 degrees, as these pairs are likely to be discriminative.

- DenseFusion [3] - Instance-level/Regression/Full pose estimator. This approach's core idea is to embed and fuse RGB values and point clouds at the per-pixel level. The argument behind this concept is that most prior works do not fully leverage RGB and depth information, where some

[3] Point Pair Features

either extract information from both channels separately and use costly post-processing steps to estimate the objects' 6DoF pose or fuse the two data sources by using the depth information as an extra channel to the RGB image. Which the authors consider as a misusage of the two sources as these data is defined in different domains. Furthermore, even methods such as [23], where the authors state that a correct use of the available information is achieved, still lacks performance under heavy occlusions as they compute global features representing image crops.

To tackle these problems, a heterogeneous architecture is proposed, where RGB data and depth information are processed individually and then densely fused at the per-pixel level, where each extracted feature will vote for a 6DoF pose.

DenseFusion is a pioneer work that inspired many other published variants of this approach.

- Methods not in [18]:

  - Meng Tian et al. [24] - Instance-level/Regression/Full pose estimator. This method uses a novel end-to-end deep network which densely extracts features from RGB-D images and estimates rotation and translation in two separate branches. The rotation estimation process is initialized with uniformly sampled rotations (referred to as rotation anchors) covering the whole SO(3) [4] space, and each one is only responsible for a local region. Given an RGB-D input, a convolutional neural network will estimate deviations from the object instance to the rotation anchors, as well as an uncertainty score for each anchor. Judging by the uncertainty scores, the method will choose an anchor as the final 3D rotation. The translation is estimated using a RANSAC-based method. The network estimates two vectors that represent the direction from two randomly selected points to the object center. Then, two 3D lines are constructed from the two pairs of point+vector, and their intersection represents a guess on the object center. The inliers are calculated, and after that, the object center is accepted as the point closest to all lines.

  - W-PoseNet [25] - Instance-level/Regression/Full pose estimator. This method aims to better discriminate global representation of objects with large variations of texture and shape. It uses the same approach as in DenseFusion [3], where color and depth information are processed differently and then fused in latter stages of the algorithm. The contribution of W-PoseNet to DenseFusion, is that it explores pixel-pair pose regression instead of just pixel-wise regression, that is, instead of fusing the depth and color information in a pixel-wise manner, W-PoseNet creates pairs of pixel features and encode them creating a new set of features. Using the encoded pixel-pair features, the 6DoF pose of the objects of interest can then be regressed.

[4] 3D rotation group

- MaskedFusion [26] - Instance-level/Regression/Full pose estimator. This method is also based on DenseFusion [3], the improvements shown by MaskedFusion are due to the introduction of object masks that eliminate non-relevant data. This method proposes a new pipeline divided into three sub-tasks that, when combined, can estimate the object's 6DoF pose. The main addition to the original DenseFusion architecture is the usage of the objects' masks obtained at the segmentation step as the input for a third neural network module.

- YOLOFF [14] - Instance-level/Classification&Regression/Full pose estimator. This method proposes a hybrid patch-based strategy. It mixes a classification and a regression convolutional neural network. At a first stage, YOLOFF uses a classification CNN to predict the 2D patches where the object might be. In the second stage, a regression CNN is used to predict the 3D key points locations. As the authors argue, by classifying the patches first, only relevant information will be used to train the regression CNN allowing it to better fit the data. Finally, on the third stage, the 6DoF pose estimation is done using the results from the previous stages.

- PointPoseNet [27] - Instance-Level/Regression/3D bounding box detector. In the first stage of this method, a 2D CNN detector is used to localize the object on the RGB image with a bounding box. The bounding box is then used to extract the corresponding region in the depth image. In the next stage, a CNN is used to predict point-wise directional vectors pointing to the 3D corners of the bounding box. Once the corner hypothesis is computed, in the third stage, a scoring mechanism is used to determine the best hypothesis that will be then used to estimate the object's 6DoF pose. A notorious feature of this method is how well it deals with occlusion.

- Shantong Sun et al. [28] - Instance-level/Regression/Full pose estimator. This method also works similar to DenseFusion [3] where the main focus is to make proper usage of both color and depth extracted features. The authors state that RGB and depth information contribute differently to the final 6DoF pose estimate and should not be simply concatenated. Here, the authors propose a selective embedding with gated fusion structure which they call SEGnet. This structure has the ability to adjust the weights of RGB features, and depth features adaptively.

- Yi Cheng et al. [29] - Instance-level/Regression/Full pose estimator. Another method that focus on how to efficiently fuse RGB and depth data. Here, the authors introduce a strategy which they call Multi-modality Correlation Learning (MMCL). This strategy is divided into two modules, IntraMCM where modality-specific discriminative features are extracted, and InterMCM, which learns to extract modality-complement features. In other words, IntraMCM searches for correlation in each channel alone and, InterMCM searches for correlation in between channels. This method also proposes two different fusion strategies for the outputs of the aforementioned modules, Fuse_V1,

Fuse_V2 and Fuse_V3. The first strategy applies the two correlation modules in parallel and, the second and third one does it sequentially. The best results were obtained with Fusion_V2.

2.1.5 Datasets

In order to train and test the methods, many image datasets have been created. Each dataset tries to represent some important scenarios and challenges that the methods may encounter on practical applications such as clutter, occlusion, many different objects, many similar objects, street views, indoor views, and light changes. In this work, the most used datasets are presented. Though the datasets will be divided into datasets for 3D BB detectors and datasets for full 6DoF pose estimators, this does not mean that they cannot be used for both types of 6DoF pose estimators. In fact, we can use whatever dataset we want to train the algorithms. The datasets are divided by the type of 6DoF pose estimator where they are used most frequently.

- Datasets for Testing and Training of 3D BB Detectors:

  - KITTI [30] - This dataset contains 14999 images of street views divided into three different categories, car, pedestrian and cyclist with a total number of 80256 labeled objects.
  - SUN RGB-D [31] - Contains 10335 RGB-D images of indoor views of house and office objects. This data also has some data collected from other datasets. There are two versions of this dataset.
  - NYU-Depth [32] - Involves 1449 images divided into 894 different categories with the main object categories being bathtub, bed, bookshelf, box, chair, counter, desk, door, dresser, garbage bin, lamp, monitor, nightstand, pillow, sink, sofa, table, tv, toilet. There are also two versions of this dataset.
  - PASCAL3D+ [33] - Has 30899 images, some of them are also taken from other datasets while the major part of them were taken from an older version of this dataset containing only 2D images that, later, are augmented with 3D annotations. The images are divided into 12 categories airplane, bicycle, boat, bottle, bus, car, chair, dining table, motorbike, sofa, train, and tv monitor.

✣ Datasets for Testing and Training of Full 6DoF Pose Estimators:

- ✣ RU-APC [34] - This dataset has 5964 frames of several different viewpoints of some objects and involving cluttered scenes.
- ✣ LINEMOD [12] - The most used dataset in recent works and also a benchmark for 6DoF pose estimation. It contains 18273 frames of objects in different viewpoints, involving cluttered scenes and texture-less objects. The objects in this dataset are ape, bench vise, cam, can, cat driller, duck, eggbox, glue, hole puncher, iron, lamp and phone.
- ✣ MULTI-I [15] - The offered challenges by this dataset are different viewpoints, cluttered scenes, texture-less objects, occlusion and multiple object instances. It contains approximately 2000 real images of 6 different objects.
- ✣ OCC [35] - The challenges of this dataset are different viewpoints, cluttered scenes, texture-less objects and severe occlusion. Occlusion is one of the hardest challenges to tackle and, this dataset focusses on that problem making it one of the hardest datasets to get good results with. The dataset contains 8916 frames of object that can be up to 70%-80% occluded.
- ✣ BIN-P [36] - It has different viewpoints, severe clutter, severe occlusion, multiple instances and bin picking challenges in it. This dataset contains the new challenge bin picking, that happens when a lot of objects are stacked in a bin. BIN-P tries to reflect the challenges found in many industrial settings. It contains 177 test images of 2 textured objects under varying viewpoints.
- ✣ T-LESS [37] - Besides the already mentioned challenges, different viewpoints, cluttered scenes, texture-less objects, occlusion and multiple object instances, this dataset contains a new challenge which is similar looking distractors, along with similar looking objects. Therefore the methods are forced to develop more discriminating selection of features which is a hard challenge. It contains 10080 frames.
- ✣ TUD-L [38] - The main focus of this dataset, is to train and test the methods to deal with different lighting conditions, hence the challenges presented in this dataset are different viewpoints and light. There are 23914 frames in it.
- ✣ TYO-L [38] - It is similar to TUD-L but it adds clutter to the scene. It contains images of 21 different object on a table-top setup with 5 different types of illumination. TYO-L has 1680 frames.
- ✣ YCB-Video [17] - It is a very recent dataset that is also one of the most used in recent works. It is also the largest dataset mentioned in this thesis. It is formed by 92 video sequences containing 21 objects, making a total of 133827 real images. It also includes 80000 synthetic images.

2.1.6 Metrics

As 6DoF object pose estimation develops through the years, several metrics were proposed to measure the whole spectrum of estimators' performance. In [18] 9 different metrics are presented. However, only three will be mentioned in this thesis: the *Average Distance* (ADD) metric, its complementary (ADD-S) that handles symmetric objects and the *Area Under Curve* (AUC) metric. Only these three metrics will be mentioned because they complement each other and are by far the most widely used metrics. The ADD and ADD-S metrics are also the only metrics used in all methods considered in this thesis.

These metrics will be presented and explained in Chapter 4 but, to aid the reader in understanding the following sections, here is a brief explanation about them. The ADD metric measures the average distance of point-pair correspondences of two point clouds, one at the ground truth pose and the other at the predicted pose. Instead of using point-pair correspondences, the ADD-S metric uses the closest points to calculate the distances. To classify whether a hypothesis is correct or not, a threshold for the ADD(-S) metrics is used. This threshold is often set as $\omega_{ADD} \leqslant k_m d$, where k_m is a chosen coefficient and d is the object's diameter. If the score is less than the threshold, then the hypothesis is accepted as correct. The AUC metric measures the area under the curve formed by the ADD(-S) metrics as a function of the aforementioned threshold.

2.1.7 Performances of the Methods on LINEMOD and YCB-Video Datasets

In this section, the results obtained on both LINEMOD and YCB-Video datasets by the methods presented in section 2.1.4 are shown. LINEMOD results are chosen because it is the only dataset for which results of all methods are available and, YCB-Video is chosen because, as it will be explained in Chapter 4, this will be the dataset used in this thesis. The results are shown in tables A.4 and A.5 in apendix A. Note that these results are published in the original papers of the methods mentioned earlier and not acquired in lab.

2.2 Point Cloud Registration

Point cloud registration, also known as point set registration, scan matching, and point cloud alignment, aims to find the spatial transformation between two point clouds of the same model. For instance, in the context of this thesis, given two point clouds representing an object from two different views or time, the goal is two align these two sets of points by estimating the 7 DoF transformation that transforms one into

the other, i.e., a scaling, a 3D translation, and a 3D rotation. Though we only need to estimate two transformations in this scope (rotation and translation), point cloud registration can deal with transformations up to 15 DoF (projective transformations). That being said, the problem can be summarized as in [1]:

- Given two sets of points A and B, and the respective putative correspondences obeying the following model:

$$b_i = s^\circ R^\circ a_i + t_i + o_i + \epsilon_i \tag{2.1}$$

Where $s^\circ > 0$, $R^\circ \in SO(3)$, and $t^\circ \in \mathbb{R}^3$ are the unknown scale, rotation, and translation, ϵ_i models the measurement noise, and o_i is zero in case of inlier correspondences and arbitrary numbers in case of outliers.

When the noise can be described by zero-mean Gaussian distribution with isotropic covariance $\sigma_i^2 I_3$ and all the correspondences are correct, the maximum likelihood estimator of the transformation can be computed solving the following nonlinear least squares problem:

$$\min_{s>0, R\in SO(3), t\in\mathbb{R}^3} \sum_{i=1}^{N} \frac{1}{{\sigma_i}^2} ||b_i - sRa_i - t||^2 \tag{2.2}$$

Where I_3 denotes an identity matrix of size three. N is the number of points in the clouds.

The problem of point cloud registration can be solved using optimization-based algorithms [1], [39], [40] and, more recently, deep learning [41],[19]. The point clouds can originate from the same-source [42], e.g., CAD models or from different sources (cross-source) [43], e.g., different RGB-D sensors. The latter case is more recent and thus less reported in the literature, as stated in [2].

3D point cloud registration has a strong connection with the 6DoF object pose estimation, as many methods still rely on this technique to solve the problem ([12],[20],[21],[14]), mainly on the refinement step. While in 6DoF pose estimation the goal is to estimate the objects absolute pose in a chosen reference frame, 3D point cloud registration focus on estimating the relative pose of an object from different time or views. This is why it is of significant importance to cover this topic in this thesis.

As in the previous section 2.1, review-based papers are used as a basis for this literature review. In this case, the attention is held into [2], where a comprehensive survey on point cloud registration is performed, and [44] where the focus is the usage of deep learning to process 3D point clouds.

2.2.1 Challenges

In this section, some of the same-source and cross-source point cloud registration challenges are summarized.

- Same-source: When dealing with point clouds captured by the same source, the challenges faced can be noise, outlier correspondences, and partial overlapping. All of these challenges lie in the fact that the data is obtained at different time or views, as the noise/error induced by the sensor is time-variant, and at different perspectives, it is not possible to capture the same views of an object, leading to only partial overlap of the point clouds.
- Cross-source: Either because the only available data was obtained with different sensors or the data is obtained in such a way to achieve better results in 3D reconstruction, as demonstrated in [43]. Many challenges arise due to the distinct intrinsic properties of the hardware. Some of these challenges can be noise and outliers, partial overlap, density difference, and scale variation.

2.2.2 Categories

In this section, the taxonomy of point cloud registration proposed in [2] is presented in Figure 2.1 and explained below. The categories and subcategories of point cloud registration are divided by the same-source and cross-source problems in the survey. However, in this thesis, only same-source categories are presented, as only one sensor was used in the YCB-Video dataset where the point cloud registration study (Chapter 5) of this thesis was carried out.

- Optimization-based registration methods: This type of methods is the most traditional way of solving the registration problem [1], [39], [40]. It is also the most accurate and assures convergence while not requiring training data. As a result of that, optimization-based methods generalize well to unseen data. By its nature, these methods are iterative and divide the solution into two main stages, correspondence search and transformation estimation.
- Feature learning methods for registration: While optimization-based methods find matches iteratively by minimizing, for instance, a distance function, feature learning methods find correspondences on the point clouds using deep learning [41],[19]. Moreover, these methods calculate the transformation matrix in a single shot. The advantages of these methods are the accuracy and robustness provided by good point-pair correspondences.

- End-to-end learning-based registration methods: In this case, the point cloud registration is fully performed by neural networks [45],[46]. It computes both correspondences and transformation matrix. These methods can be powerful in terms of accuracy and generalization and are mathematically simpler than optimization-based methods, making them easier to implement and set up. However, because of that, these are also more sensitive to noise and density differences.

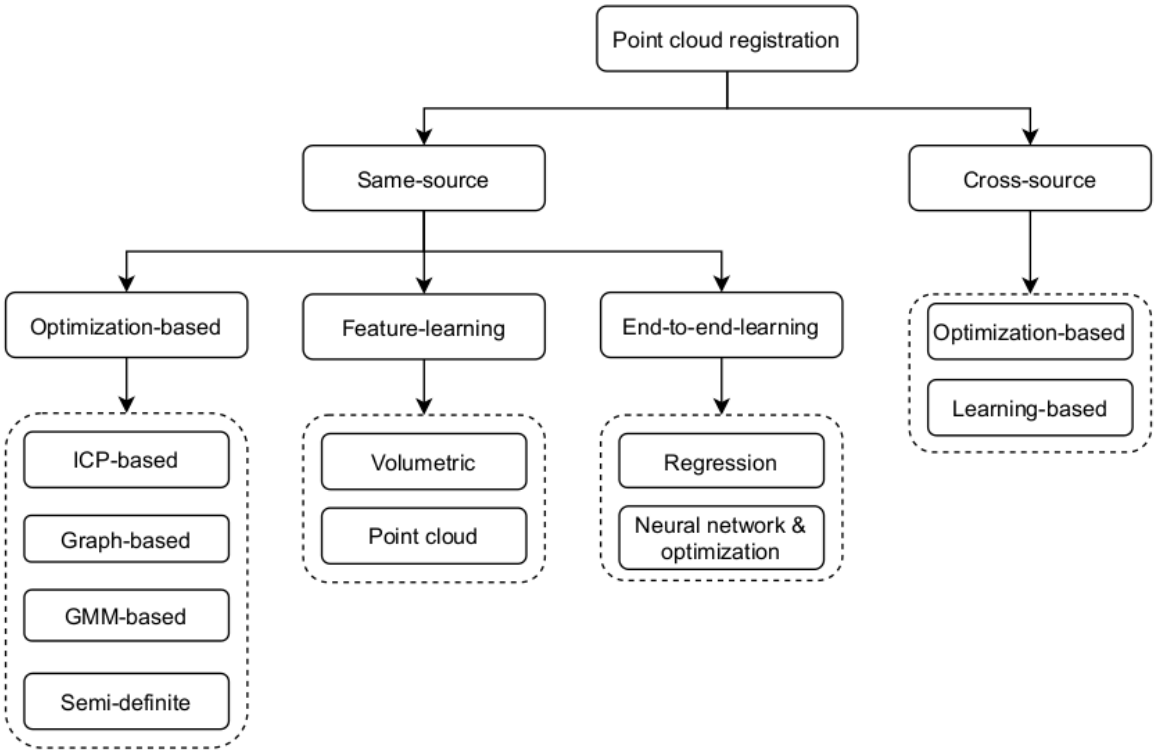

Figure 2.1: Taxonomy of point cloud registration. [2]

2.2.3 Subcategories

As it will be explained further in Chapter 5, the experiments with point cloud registration in this thesis are conducted on a specific point cloud registration method, TEASER++ [1]. This method is optimization-based and, in addition to that, some feature-learning methods are also mentioned in the discussion. Bearing that in mind, only the subcategories of these two types of methods will be presented.

- Optimization-based:

  - ICP-based registration: ICP stands for *Iterative Closest Point* and is is arguably the most widely used method for rigid point cloud registration. It was originally proposed in [47] and has many variations

[48], [40], [49]. This method consists of two steps: robust point-pair correspondence and accurate transformation estimation. The matches are firstly calculated as the closest points of the two input point clouds, and, given an initial transformation, a distance metric can be computed. The method will iterate until convergence. The main goal of this algorithm is to search for a transformation matrix that minimizes the distance metric.

- Graph-based registration: In this approach, the correspondence search is formulated as a graph matching problem [50], where the point clouds are modeled as graphs and, by finding similarities in both vertexes and edges of the graphs, the matches can be computed. This process is iterative and tries to maximize an objective function that outputs a score measuring how good the similarities are.

- GMM-based registration: GMM stands for *Gaussian Mixture Model*, which in statistics, is a technique used to find sub-populations in a larger population. A point cloud can also be viewed as a population of some sort, thus building the bridge between statistics and registration. This approach's key idea is to represent discrete sets of points with continuous density functions [51], namely Gaussian functions. By modeling the problem this way, the point clouds can be aligned with respect to their distributions.

- Semi-definite registration: Methods based on this technique focus into developing good approximation models in order to facilitate the correspondence search process. To develop a good approximation model, one can start by building a square matrix with binary values, where a match is represented with a "1", and "0" otherwise. This matrix is expected to be symmetric, and the sum of each row and column should be no larger than one. By working on this matrix and considering its intrinsic properties, some methods [52], [1], find good and efficient ways to find the input point clouds' correspondences.

- Feature Learning:

Feature learning methods can either work with volumetric data or directly with point clouds, where the only difference between these two approaches is that some methods [41], [53], pre-process the point clouds and represent them as volumetric data. As demonstrated in [41], by representing the point clouds in this manner, it is possible to achieve better results in feature-learning-based registration.

2.2.4 Methods for Point Cloud Registration

Considering that this thesis is mainly focused on 6DoF pose estimation and not 3D point cloud registration, in this Chapter, only the methods used and mentioned in Chapter 5 will be presented. The chosen method for point cloud registration is TEASER++ [1], which is very recent and reports excellent results in the presence of large amounts of noise and outliers. In fact, this method performance claims were one of the primary motivations for this thesis inherent studies. As an extra, all the code required to implement this method is open source, making this algorithm very suitable for this work.

Other methods are also mentioned in this thesis because TEASER itself does not perform correspondence match, thus requiring a complementary method to carry out this step. By design, TEASER can solve the registration problem without matches, with the cost of worse accuracy and slower inference time. Hence, a good mechanism is needed to match the points first.

The following list presents the highlighted methods in this thesis and a short explanation of how these work.

- TEASER++ [1]: It is optimization-based and can be included in the semidefinite subcategory. TEASER stands for *Truncated least-squares Estimation And SEmidefinite Relaxation*. It is truncated because the method is able to discard outlier point-pair correspondences from the transformation approximation. Semidefinite relaxation is done at the rotation estimation step. As the authors state, finding a least-squares solution for the transformation is hard to solve when compared with previous methods like RANSAC-based methods. Hence, the authors also present a way of decoupling the scale, rotation, and translation estimation to simplify this problem. TEASER++ shows great results under meager rates of inlier correspondences and in terms of computation time. It is fair to say that this method is one of the most accurate and robust methods in the point cloud registration field.

- 6-PACK [19]: This method was not developed as a point cloud registration algorithm but as a 6DoF pose tracker. Even though this work's authors encapsulate the 6DoF pose estimation and tracking as a whole, their main contribution is their keypoint detection approach, making this method relevant for the topic in question. This model uses a dissected DenseFusion network to compute features from the RGB image and depth maps. Then, they use these features to compute key points using an attention mechanism over a grid of anchor points generated around the object's predicted pose. Each anchor summarizes the points around them. By doing so, this model is able to find a coarse centroid of the object, which will guide the following search for key points. Another network then learns the key points in an unsupervised manner. From the point cloud registration point of view,

6-PACK can be seen as a feature-learning-based method, where the learned key points can be used to align two sets of points.

- 3DSmoothNet [41]: Is a feature-learning-based registration method where the input point clouds are first represented as volumetric data. The proposed workflow is divided into four steps: Given two raw point clouds, a local reference frame (LRF) is computed on randomly selected points' spherical neighborhood. The neighborhoods are then transformed into their canonical representations, voxelized and smoothed into a smoothed density value (SDV) representation. In the end, the local feature descriptors are computed with the so-called 3DSmoothNet. The authors of this work state that, by preprocessing the point clouds this way, it is easier to train the neural networks on the later steps and save network capacity to learn highly descriptive features. When it was launched back in 2019, this method outperformed the state of the art in both 3DMatch and ETH datasets. It is suggested in [1] the deployment of this method to compute point-pair correspondences, and this is why it is considered in this thesis.

- FPFH [42]: *Fast Point Feature Histograms* is a variation of [54]/[55]. This method focuses on keypoint detection and does it by iteratively searching for each point's k-nearest neighbors in the point cloud and by computing the surface normals of each neighborhood using Sigular Value Decomposition (SVD). The surface normals are then used as feature descriptors. If there is a rule to detect and describe certain points in the input point clouds, these points can be matched and later used for registration. This method is also mentioned in [1], and that is the reason why it is considered in this research.

2.2.5 Point Cloud Registration on RGB-D Data

While only in very specific scenarios, such as medical applications, a complete point cloud of an object can be obtained, in robotics, the only way of obtaining depth information is through RGB-D or depth-only sensors that can only see what is in front of them, i.e., the point clouds obtained with these sensors only represent the surfaces of the objects facing them.

This phenomenon is called self-occlusion and is somewhat overlooked in the literature, as very few methods [56], [57], [58],[59] are designed to deal mainly with this type of data. Aligning RGB-D data is more challenging, as most of the possible correspondences are outliers. Moreover, such data contains a lot of noise and error that the sensor itself introduces, and due to its limited resolution, it is almost impossible to get depth information of the same point in space in two different images.

Having said that, it is essential to highlight that this problem is acknowledged in this research. Accordingly, one of the key contributions of the study performed in Chapter 5 is also to evaluate how TEASER++ deals with self-occlusion.

2.3 RGB-D Sensors

An RGB-D sensor is the combination of a typical RGB camera with a depth sensor. These devices were initially introduced to the market as a home entertainment gadget [60]. However, as these developed through the years, RGB-D sensors rapidly found their way into research, and industrial applications such as indoor robotics [61], medical [62], agricultural [63], 3D scene reconstruction [64], and self-driving vehicles [65]. In this thesis, three different RGB-D sensor technologies are highlighted: Coded light, active stereoscopy/stereo vision, and time-of-flight (ToF). These technologies will be explained in the following sections, joined by some of the devices benefiting from them.

2.3.1 Coded Light based

Sensors that use this technique are able to compute depth with the help of a light/infrared projector. A known pattern is projected onto the scene and, by measuring how it deforms, it is possible to infer depth. This technique is considered indirect depth estimation. The first RGB-D sensors available to the public were based on this technique (first-generation Kinect). Some sensors that use this technology are the first generation kinect from xbox 360 [16], Intel Realsense SR305 [66] and Orbbec Astra [67].

Figure 2.2: Coded Light based RGB-D sensors. *Kinect for Xbox 360 on the left, Intel RealSense SR305 on the center, Orbbec Astra on the right.*

2.3.2 Stereo Vision based

Stereo vision is a technique where the scene is captured with two cameras. By measuring the disparities of the two obtained images, it is possible to calculate depth. Taking the pin-hole camera model into account, and if we overlap the two images, closer features in the scene should appear with a wider interval in between them than with the farther ones.

This method is also considered an indirect manner of measuring depth, and its performance relies on the quality of the feature extraction algorithm used. Hence, a variation of this technique is to use a light/infrared projector to create synthetic features in the images, referred to as Active Stereo.

Some sensors that use this technology are the Intel RealSense D415 [68], Orbbec Astra Stereo S U3 [69] and OAK_D [70].

Figure 2.3: Stereo Vision based RGB-D sensors. *Intel RealSense D415 on the left, Orbbec Astra Stereo S U3 on the center, OAK-D on the right.*

2.3.3 LiDAR based

Light Detection And Ranging is the only of the three mentioned technologies that measures depth directly. It is also the least affordable and the more recent to become available to the consumer-level market. Sensors that use this method come in an embedded package, including an RGB sensor and a MEMS [5] that emits a signal of one or more specific wavelengths into the scene and, by measuring the ToF (Time of Flight), it can compute depth. In other words, it measures the time in between light-emitting and light-reception.

Though in this research, only the MEMS version of this sensor is mentioned, there are other forms of this device, such as the ones found in autonomous vehicles, where a different type of LiDAR sensor is deployed.

As it will be shown in Chapter 3, RGB-D sensors based on this technique can be the most accurate and precise. Some examples of these devices are the Intel RealSense L515 [71], Microsoft Kinect 2 [72], and ASUS Xtion 2 [73].

Figure 2.4: LiDAR based RGB-D sensors. *Intel RealSense L515 on the left, Kinect 2 on the center, ASUS Xtion 2 on the right.*

[5] Microelectromechanical System

2.3.4 Experimental Evaluation

Depth cameras and RGB-D cameras have been analyzed and compared in many different ways. In [74] bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments of ten depth cameras were experimentally analyzed and compared. A comparative study on structured light and time-of-flight based Kinect cameras is elaborated in [75]. In [76] depth cameras were compared considering medical applications and their specific requirements. [77], [78], and [79] make a comparison of RGB-D sensors in the scope of medical, agricultural and robotic applications, respectively. In [80] several RGB-D sensors are analyzed and compared based on controlled displacements, with precision and accuracy evaluations.

2.4 Conclusion

In this Chapter, the state of the art relating to 6DoF pose estimation, 3D point cloud registration, and RGB-D sensors was presented. Since this thesis mainly focuses on 6DoF pose estimation, this topic was investigated more thoroughly than the others, where the more relevant methods, datasets, metrics, and even the methods performances on two benchmark datasets were presented. Concerning 3D point cloud registration, only the methods considered in this thesis were presented, but all the fundamentals of this field were covered. In the end, the fundamentals of RGB-D sensing were also presented, where the three main techniques for RGB-D detection were covered, joined by three relevant devices based on each technique.

Chapter 3

Intel RealSense SR305, D415 and L515: Experimental Evaluation and Comparison of Depth Estimation

3.1 Introduction

Consumer-level RGB-D cameras are affordable, small, and portable. These are some of the main features that make these types of sensors very suitable tools for research and industrial applications, ranging from practical applications such as 3D reconstruction, 6DoF pose estimation, to augmented reality, and many more [81]. For many applications, it is essential to know how accurate and precise an RGB-D camera is to understand which sensor best suits the specific application [82]. In this thesis, three models of RGB-D cameras from Intel are compared, aiming to find the best fit for the 6DoF object pose estimation problem.

The sensors are the RealSense SR305, D415, and L515. Each sensor uses different methods to calculate depth. SR305 uses coded light, where a known pattern is projected into the scene and, by evaluating how this pattern deforms, depth information is computed. D415 uses stereo vision technology, capturing the scene with two images and, by computing the disparity on the two images, depth can be retrieved. Finally, the L515 that measures time-of-flight, i.e., this sensor calculates depth by measuring the delay between light emission and light reception.

Several different approaches can be used to evaluate depth sensors [83],[84]. In this case, the focus is on accuracy and repeatability. For this purpose, the cameras were evaluated using depth images of 3D

planes at several distances, but whose ground truth position and orientation were not used as these are unknown. Accuracy was measured in terms of point-to-plane distance, and precision was measured as the repeatability of 3D model reconstruction, i.e., the standard deviation of the parameters of the estimated 3D model (in this case, a 3D plane). The average number of depth points per image where the cameras failed to calculate depth (and their standard deviation), and the number of outliers [1] per image are also calculated. Moreover, it is proposed the application of directional statistics [85] on the planes' normal vectors as a repeatability measurement.

3.2 Methodology

3.2.1 Materials

As aforementioned, the sensors used in this evaluation employ different depth estimation principles, which yields information about the sensor's performance and how these technologies compare to each other (for the specific criteria used).

The SR305 uses coded light, the D415 uses stereo vision, and the L515 uses LIDAR. The camera specifications are presented in table 3.1. The three cameras were mounted on a standard tripod.

To ensure constant illumination conditions, a LED ring [86] was used as the only room illumination source.

Sensor	**SR305**	**D415**	**L515**
Depth	640x480	1280x720	1024x768
Color	1920x1080	1920x1080	1920x1080
Range	[0.2 1.5]	[0.3 10]	[0.25 9]

Table 3.1: Sensors resolution (px*px) and range (m).

Note that the values on table 3.1 are upper bounds, meaning that the specifications may vary for different sensors configurations. It is also important to mention that the D415 range may vary with the light conditions.

[1] Points for which the depth was outside an interval

3.2.2 Experimental Setup

Each camera was mounted on a tripod and placed at a distance d of a wall. The wall is white and covers all the field of view of the cameras. The optical axes of the sensors are approximately perpendicular to the wall. Placed above the camera is the light source described above. The light source points to the wall in the same direction as the camera. For practical reasons, the light source is slightly behind the camera so that the camera does not interfere with the light. A laptop is placed behind the camera, where the camera software is executed and where the images are stored. All the experiments took place at night, avoiding any unwanted daylight. Hence, the room light was kept constant between experiments and always sourced by the same element.

The camera, light source, and laptop were placed on top of a structure. Here, we want everything to be high relative to the ground to ensure that the sensors captured neither floor nor ceiling. For each distance at which the cameras were placed, 100 images were acquired. The distances for which both D415 and L515 were tested are $0.5m$, $0.75m$, $1m$, $1.25m$, $1.5m$, $1.75m$, $2m$, $2.5m$, $3m$, and $3.5m$. The furthest distance was the maximum distance for which neither floor nor ceiling appeared on the images. The SR305 was tested at $0.3m$, $0.4m$, $0.5m$, $0.75m$, $1m$, $1.25m$, and $1.5m$. In this case, the furthest distance is the maximum specified range for the SR305 sensor.

The experiments started at the closest distance. The sensors were switched one after the other sequentially. After all the images were obtained at that distance, the structure was moved away from the wall by the aforementioned intervals. The structure moved approximately perpendicularly to the wall.

For the D415 and the L515 sensors, custom configurations were used. For the SR305, the default configuration was used. These configurations were the same as those of table 3.1.

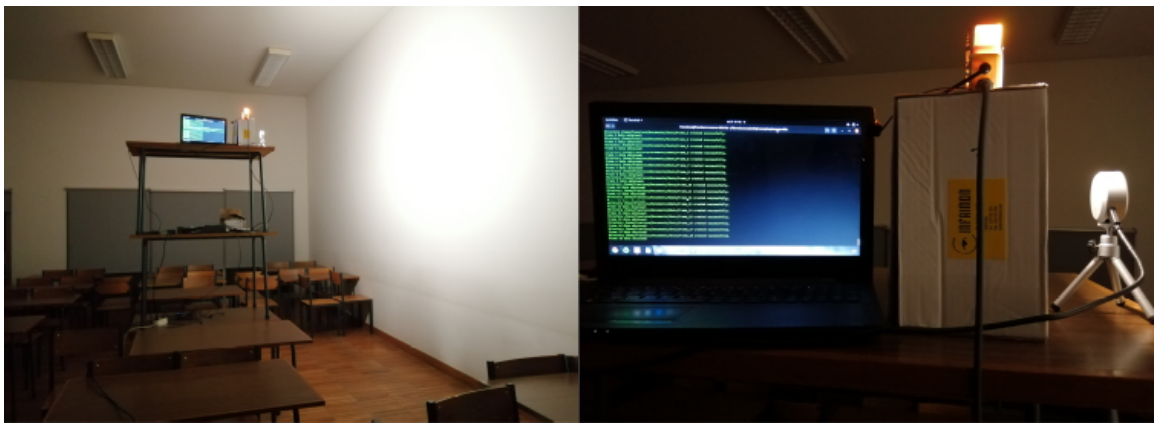

Figure 3.1: Intel RealSense SR305, D415 and L515: Experimental Evaluation and Comparison of Depth Estimation - Experimental setup.

3.2.3 Software

To deal with the sensors, the Intel RealSense SDK 2.0 was used. The Intel RealSense Viewer application was used to check the sensors' behavior right before each execution, and to check for the optical axis and distance direction. All the other tasks were executed using custom code and the librealsense2 library. These tasks included both image acquisition and storing, camera configuration, and point cloud generation. This part of the work was executed using Ubuntu. All the statistical evaluation was performed in MatLab, on Windows 10.

3.2.4 Experimental Evaluation

✣ Performance

The performance of the sensors was measured in two ways. First, the average number of points for which the sensor failed to measure depth and the standard deviation of the same number of points was calculated. Then, the same was done for outliers.

Whenever the Intel RealSense SDK 2.0 and camera fail to measure the depth at some point, the corresponding depth is set to zero. Hence, all we do here is to count pixels in the depth image with a depth equal to zero.

Depth values also contain outliers. Outliers can be defined in several ways. In this case, an outlier is considered as every point with a depth value differing $10cm$ from the expected distance, given the specific geometric configuration and setup.

As described in the Intel RealSense D415 product DataSheet [87], the D415 sensor has an invalid depth band, which is a region in the depth image for which depth cannot be computed.

The coordinate frame of the left camera is used as the reference coordinate frame for the stereo camera. The left and right cameras have the same field of view. However, due to their relative displacement, there is an area in the left image for which it is impossible to compute disparities since the corresponding 3D volume is not visible in the right camera. It results in a non-overlap region of the left and right cameras for which it is impossible to measure depth. This region appears in the image's leftmost area and is illustrated in figure 3.2. The total number of pixels in the invalid depth band can be calculated in pixels as follows:

$$InvalidDepthBand = \frac{VRES * HRES * B}{2 * Z * tan(\frac{HFOV}{2})} \tag{3.1}$$

Where $VRES$ and $HRES$ stand for vertical and horizontal resolution respectively ($720px$ and $1280px$), B is the baseline $\rightarrow$ 55mm, Z is the distance of scene from the depth module $\rightarrow d$ and $HFOV$ is the horizontal field of view $\rightarrow$ 64°.

Bearing that in mind, the pixels in the invalid depth band were ignored in the calculations.

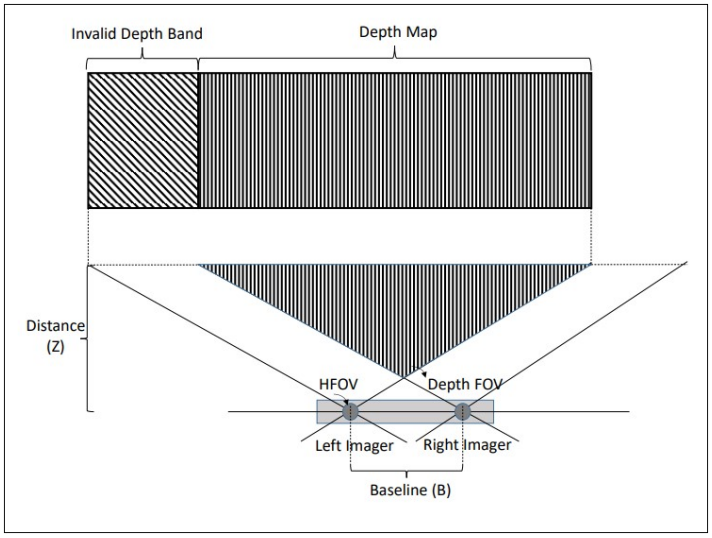

Figure 3.2: Invalid Depth Band.

✣ Plane Fitting

Point clouds were first obtained using the depth data, the image pixel coordinates, and the camera intrinsics. This is possible because we have depth information, letting the coordinate z be equal to the measured depth at that point, i.e., $z = d_m$ the following equations can be applied:

$$x = z * \frac{u - pp_x}{f_x} \qquad (3.2) \qquad\qquad y = z * \frac{v - pp_y}{f_y} \qquad (3.3)$$

Where (u, v) are the pixel coordinates, (pp_x, pp_y) are the coordinates of the principal point, f_x and f_y are the focal lengths in pixel units.

The point clouds correspond to a wall. Thus it is possible to fit a plane to the data.

Since the outliers are handled manually, the plane equation's estimation is performed using standard least-squares regression, employing the singular value decomposition, instead of robust approaches such as RANSAC.

The model we want to be regressed to the point clouds is the general form of the plane equation:

$$x * n_x + y * n_y + z * n_z - d = 0 \tag{3.4}$$

Where (n_x, n_y, n_z) stands for the unit normal, and d stands for distance from the plane to the origin.

If we now build a $n * 4$ matrix from the point clouds with n points, which will be denoted as matrix P. We can rewrite equation 3.4:

$$0 = \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 & -1 \\ x_2 & y_2 & z_2 & -1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & -1 \end{bmatrix}}_{P} \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} \tag{3.5}$$

By computing the singular value decomposition on matrix P as:

$$P = U * \Sigma * V' \tag{3.6}$$

We can now use the values of the column of matrix V that corresponds to the smallest eigenvalue in matrix Σ, as the parameters n_x^*, n_y^*, n_z^* and d^* of the plane that fit that point cloud. Then, the plane's normal vector is normalized, which will become handy in further calculations and to recover the true distance in meters of the plane from the sensor.

✣ Accuracy

For the accuracy analysis, point-to-plane distance is computed. For each point of the point cloud, the fitted plane equation is used to compute the errors, i.e., the point-to-plane distance. Then, the average root mean square error for each set of 100 images is computed as an accuracy measurement. For the sake of comparison, this computation is performed for two different threshold values of outlier rejection ($\pm 10cm$ and $\pm 35cm$), another one where all points with measured depth are used.

✣ Precision

In this experiment, precision is measured as per image plane consistency, i.e., how the plane model changes in between images of the same sensor at the same distance. As neither the scene nor the sensor change

while taking the pictures, we could expect the models to be the exact same if we had an ideal sensor. Thus, by measuring the standard deviation of the plane model parameters in between images, we might be able to better understand how consistent the sensors are with their measurements and how this consistency varies with the distance.

Additionally, the plane's normal vector is transformed into spherical coordinates, where analysis in terms of directional statistics can be performed, as all the normals are distributed on a spherical surface. Specifically, the circular mean and standard deviation of angles θ and ϕ, and the spherical variance of the normal vectors. Since $\vec{n_i}$ is unitary, its norm ρ is 1.

Let θ_i and ϕ_i be the azimuth and polar angles of $\vec{n_i}$:

$$\theta_i = \arctan \frac{n_{y_i}}{n_{x_i}} \tag{3.7}$$

$$\phi_i = \arctan \frac{\sqrt{n_{x_i}^2 + n_{y_i}^2}}{n_{z_i}} \tag{3.8}$$

As in [88], the circular mean of the angles above can be computed as follows:

$$\overline{\theta} = \arctan \frac{\sum_{i=1}^{n} \sin \theta_i}{\sum_{i=1}^{n} \cos \theta_i} \tag{3.9}$$

$$\overline{\phi} = \arctan \frac{\sum_{i=1}^{n} \sin \phi_i}{\sum_{i=1}^{n} \cos \phi_i} \tag{3.10}$$

Now, to show how the spherical variance is computed, we need to introduce vector $\overline{\vec{n}}$, which is the vector whose components are the mean of each component of $\vec{n}$. If we now compute the norm of $\overline{\vec{n}}$ and call it $\overline{R}$, the spherical variance is calculated as follows:

$$V = 1 - \overline{R} \tag{3.11}$$

3.3 Results

3.3.1 Performance Analysis

In table 3.2 and figure 3.3, the results for the average number of failed points and the standard deviation of the number of failed points per image [2] are presented. As it can be verified, the L515 sensor outperforms both D415 and SR305. This sensor not only showed to be capable of estimating more depth data relative to its resolution, but the results also show that that number (of failed points) remains almost constant up until 2 meters of distance. This can be explained by the fact that this sensor uses LiDAR technology, which is more robust than the stereo vision and coded light approaches since LiDAR directly computes depth.

distance	D415	L515	SR305
0,3	—	—	1.3967% ± 0,0397%
0,4	—	—	0.6062% ± 0,0198%
0,5	0,2801% ± 0,0248%	0,0020% ± 0,0023%	1.6930% ± 0,0415%
0,75	0,5099% ± 0,0224%	0,0001% ± 0,0003%	6.0734% ± 0,0783%
1	0,5414% ± 0,0195%	0,0023% ± 0,0050%	25.8181% ± 0,1282%
1,25	0,6415% ± 0,0211%	0,0001% ± 0,0004%	56.3898% ± 0,1886%
1,5	0,3025% ± 0,0329%	0,0014% ± 0,0033%	84.2698% ± 0,2714%
1,75	0,7472% ± 0,0458%	0,0017% ± 0,0025%	—
2	1,1202% ± 0,0319%	0,0320% ± 0,0100%	—
2,5	0,7945% ± 0,0335%	0,1648% ± 0,0278%	—
3	0,7660% ± 0,0378%	0,8263% ± 0,0570%	—
3,5	0,8644% ± 0,0585%	0,5456% ± 0,0155%	—

Table 3.2: Average and standard deviation of failed points ratio per image by distance in meters.

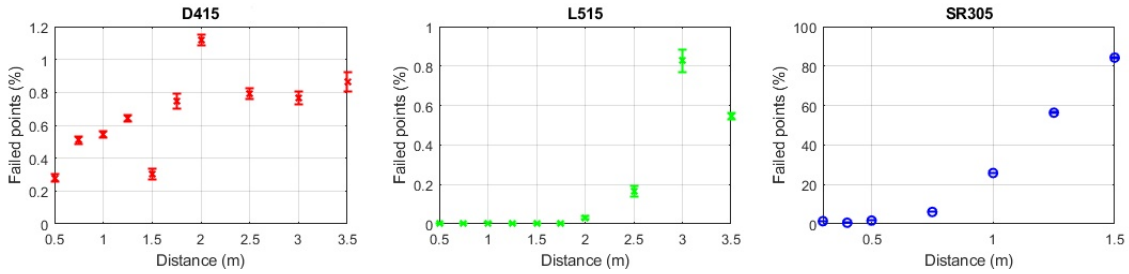

Figure 3.3: Failed Points - D415 vs L515 vs SR305.

On the other hand, the D415 sensor shows much better results than the SR305.

A relevant detail of this performance measurement is that the standard deviation of the number of failed depth points for the D415 is not strongly dependent with the distance, whereas that does not seem to be the case for the L515 after $1,75m$.

[2] points where the sensor could not compute depth

Table 3.3 and figure 3.4 include the results for the number of outliers. Just as it happened in terms of the average number of failed depth points, the data for the D415 camera show an increase in the average number of outliers as the distance increases. On the other hand, the number of outliers for the SR305 seems to decrease with increasing distances. These results are quite different from those presented in table 3.2. The main reason for these results is probably because the SR305 camera is estimating a relatively small number of depth points at higher distances, therefore decreasing the probability of outlier occurrences.

distance	D415	L515	SR305
0,3	—	—	0,1560% ± 0,0004%
0,4	—	—	0,0933% ± 0,0018%
0,5	0,1141% ± 0,0209%	0,0968% ± 0,0000%	0,0000% ± 0,0000%
0,75	0,1112% ± 0,0193%	0,0753% ± 0,0034%	0,0081% ± 0,0105%
1	0,0667% ± 0,0139%	0,0968% ± 1,2715%	3,9062% ± 0,0003%
1,25	0,2809% ± 0,0226%	0,0968% ± 0,0000%	0,0032% ± 0,0046%
1,5	0,5387% ± 0,0380%	0,0968% ± 0,0000%	0,0063% ± 0,0057%
1,75	0,2568% ± 0,0431%	0,0813% ± 0,0025%	—
2	0,0558% ± 0,0168%	0,0957% ± 0,0008%	—
2,5	0,8361% ± 0,1892%	0,0882% ± 0,0018%	—
3	3,9436% ± 0,5601%	0,0839% ± 0,0027%	—
3,5	10,3307% ± 0,7226%	0,0609% ± 0,0001%	—

Table 3.3: Average and standard deviation of outliers $\pm 10cm$ ratio per image by distance in meters.

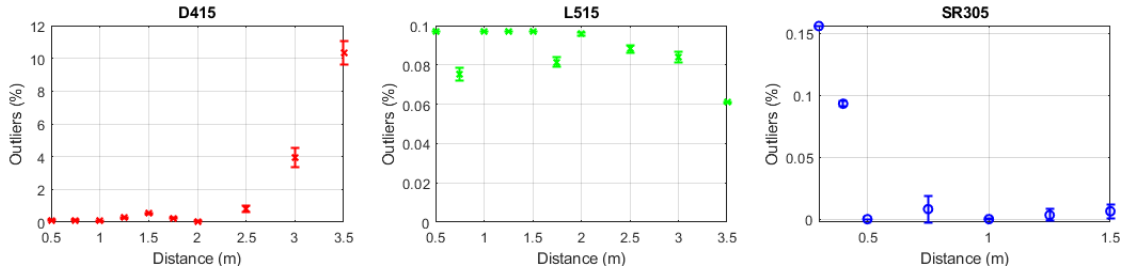

Figure 3.4: Outliers - D415 vs L515 vs SR305.

To better illustrate this, in figure 3.5 a sample image taken with the SR305 sensor at 1.5 meters of distance is shown, where it is notorious the small amount of points for which this sensor is measuring depth, the dark region represents points where the depth was not computed.

On the other hand, for the case of the L515, the number of outliers is essentially independent of the distance. Even though there is some fluctuation in the numbers, the variation is relatively small. Considering the standard deviation of the number of outliers per image obtained with the L515 at 0.75, 1.5, and 1.75 meters, we can see that it is zero, meaning that the total number of outliers per image stayed constant over the 100 images. This led us to determine the pixels where L515 generated outliers. It was found that the miscalculated points always correspond to the same pixels from the image's leftmost column. It is believed that this may be happening due to an issue with the camera used in the experiments, which

requires further investigation.

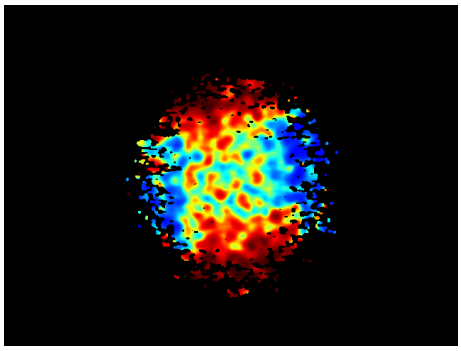

Figure 3.5: SR305 depth image at 1,5m.

3.3.2 Accuracy Analysis

The results of the accuracy study are represented on table A.1 and in figure 3.6, where the point-to-plane RMSE [3] distances for the three cameras are plotted, taking into account only points whose distances from the expected distance are within $10cm$.

Again, the sensor that achieves the best results is the L515. It not only has the lowest average root mean square error per image, but it also shows to be the least sensitive to distance. It should be mentioned that the image acquisition conditions for the L515 were close to being optimal since the images were acquired indoors, without daylight illumination.

In the case of camera D415, its RMSE follows a smooth increasing pattern as it goes further from the wall. On the other hand, the RMSE for the SR305 camera does not vary as smoothly with distance. The RMSE values for this sensor increase until $0.75m$ and then start to decrease until $1m$. In fact, 1 meter is the distance for which the SR305 is optimized [66], therefore one should expect this sensor to work better within this range.

3.3.3 Precision Analysis

The results show that the camera L515 is significantly more consistent than the other sensors. The results from tables A.2 and A.3 show the L515 to be more precise in terms of 3D estimation. It is noticeable how this sensor seems to be very consistent between pictures and for different distances.

[3] Root Mean Square Error

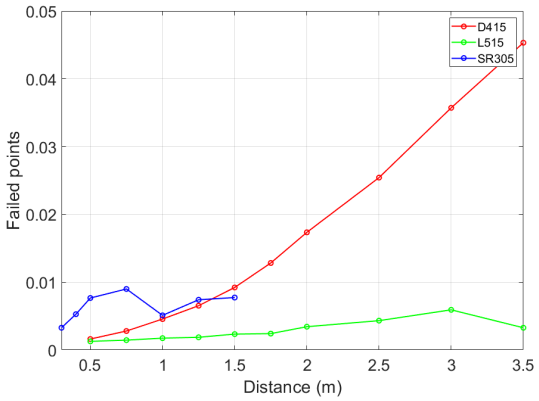

Figure 3.6: Point-to-plane RMSE D415 vs L515 vs SR305 - $\pm 10cm$.

In table A.3, the directional statistics results are presented. The values for the angle θ frequently change in a non-systematic way. This happens because, as ϕ gets closer to 90°, the components n_x and n_y of the normal vector get closer to zero, which will lead to large variations of the angle θ. For this reason, the azimuth calculations are omitted.

The spherical variation behaves just as expected, showing again that the L515 sensor is the most stable and that the measurements from the other two are more distance sensitive.

For ease of comprehension of the precision results, the standard deviation of parameter d of the plane for all cameras at all distances and the spherical variation are plotted in figures 3.7 and 3.8.

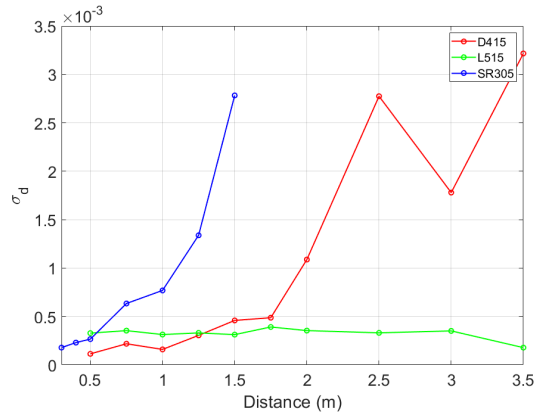

Figure 3.7: Parameter d standard deviation.

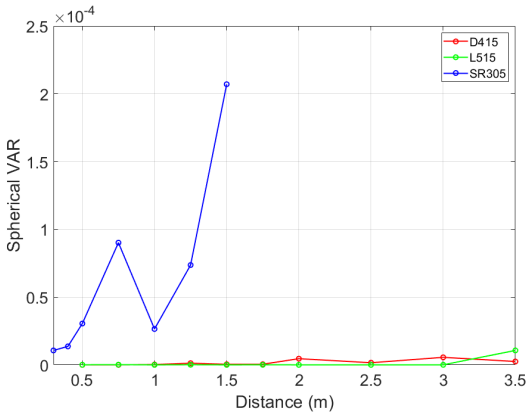

Figure 3.8: Spherical Variation.

3.4 Conclusion

In this study, a set of experiments were performed and described to compare the depth estimation performance of three RGB-D cameras from Intel, namely the SR305, the D415, and the L515. In general, the results show that the L515 is more accurate and precise than the other two while also providing more stable and consistent measurements in the specific environmental conditions of the experiments (indoors with controlled and stable illumination). Therefore, it is fair to say that the L515 is the best fit for the 6DoF pose estimation field.

Chapter 4

Implementations

4.1 Inroduction

Looking forward to the following Chapters of this research and its inherent studies, a 6DoF object pose estimator is needed. To solve this, two different possibilities are considered, to get third party software, or to implement the code. It was decided to implement the methods since a specific implementation is advantageous in the sense that it not only provides much more flexibility on the studies to perform, but it allows for easy introduction of changes in the model.

That being said, the question of what method to use remains. Desirably, all methods mentioned in Chapter 2 would be implemented, tested, and compared, to determine the model that best adjust to the problem addressed in this dissertation. However, considering the complexity of machine learning modeling, and that training all of the methods is highly time-consuming, this is not feasible. Therefore, one can only rely on the available studies on the literature and the results shown in table A.5, because the focus here is the YCB-Video dataset. Another criterion is ease of implementation and available online support, as this is an important aspect to consider when implementing such tools. Hence, the chosen method for this research is DenseFusion [3].

It also happens that DenseFusion not only has online support, but there are also several recently published works inspired on this architecture [25], [26], [28].

In addition, DenseFusion is currently one of the most accurate 6DoF pose estimators. Also the architecture is not significantly complex.

Therefore, the methodolodgy of DenseFusion is presented, including the implementation details as well

as the training and testing methodology.

A proposed change to the to the original architecture is also presented.

In the end, the performance of the implemented model on the YCB-Video dataset is reported.

4.2 DenseFusion

DenseFusion is an end-to-end deep learning approach to the 6DoF object pose estimation problem, where the main focus is to fully leverage RGB and Depth data. Accordingly, the authors of [3] propose a deep neural network architecture where the color and depth images are processed independently in the first stages of the model and densely fused later on. The fusion of the extracted features from the two types of data is done at the per-pixel level, a concept that the authors claim to be more robust against heavy occlusions. In the end, each densely fused feature will vote for a 6DoF pose of the object, joined by a confidence score calculated by the network itself. The pose with the highest confidence is accepted as the 6DoF pose of the object.

4.2.1 Architecture

The architecture of DenseFusion is shown in Figure 7.1

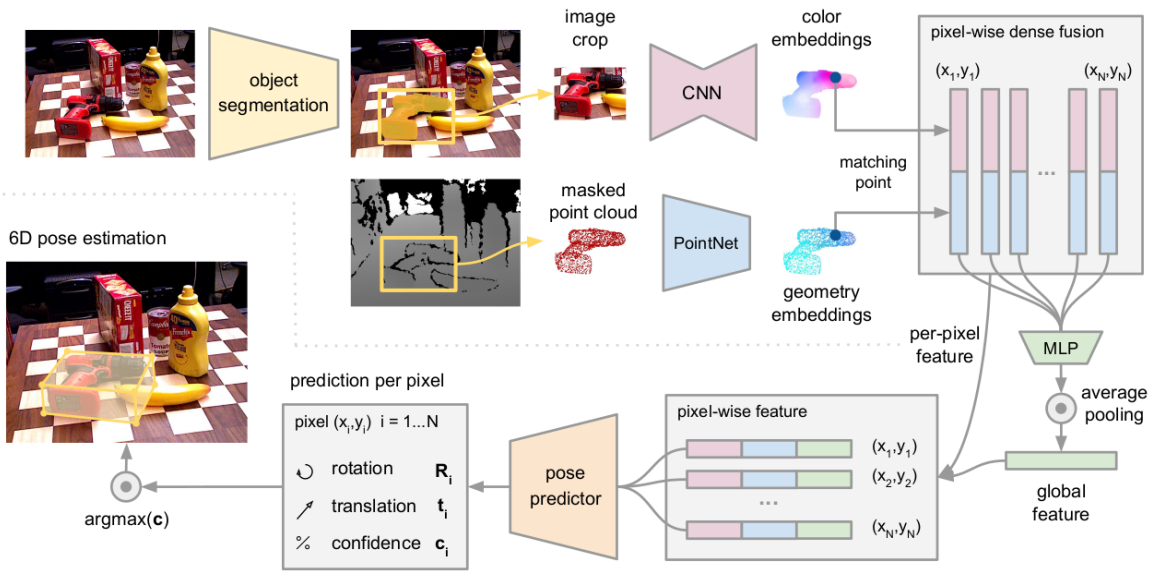

Figure 4.1: Overview of DenseFusion's 6DoF pose estimation model. [3]

- Object segmentation block: This is a process often called image segmentation, semantic segmentation, or semantic labeling. It is the first stage of the model, and it is where the image is interpreted, where each object is recognized and localized in the image. It takes only the RGB image and labels every pixel with the corresponding object class.

  The segmentation network used in this approach is the same as the one proposed in [3], which is proposed for the 6DoF object pose estimation problem in [17], and inspired by [89]. It is a convolutional neural network arranged in an encoder-decoder configuration. In Figure 4.2, this network is illustrated.

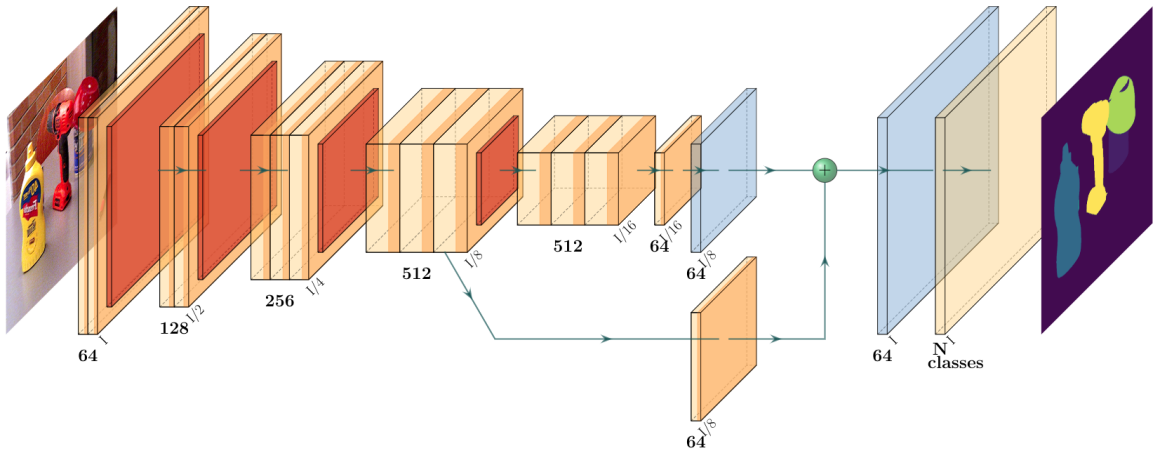

Figure 4.2: Semantic labeling network. *Yellow boxes are convolutions. Blue boxes are deconvolutions. Red boxes are max pooling. Dark yellow bands represent ReLU activation.*

The encoder part of the network is divided into five convolutional blocks with max-pooling layers between them, reducing the image resolution by 2 in each of them. This is where the image features are extracted. In total, the feature extraction stage has 13 convolutional layers.

The semantic labeling branch is the decoder part that takes two feature maps with channel dimension 512 generated by the feature extraction stage as inputs. The resolutions of the feature maps are $\frac{1}{8}$ and $\frac{1}{16}$ of the original image resolution. The channel dimension is first reduced from 512 to 64. Then, the smaller feature map is up-sampled with a deconvolutional layer in order to match the resolution of the other feature map ($\frac{1}{8}$ of the original image size). The two feature maps are summed up and then fed to another deconvolutional layer that will increase its resolution by 8, matching the original image's size. Finally, a last convolutional layer is used to classify each pixel, i.e., to generate an image with channel dimension $N_{classes}$, where each class represents a different object. If a pixel belongs to the background, then it is labeled with class "0".

- Image crop and point cloud masking: This is the step where the images are prepared to be fed to the DenseFusion pipeline. It takes the output of the segmentation network and the RGB-D images as inputs, and outputs two batches, one containing the image crops stacked all together, and another one containing the masked point clouds also stacked. This procedure is illustrated in Figure 4.3.

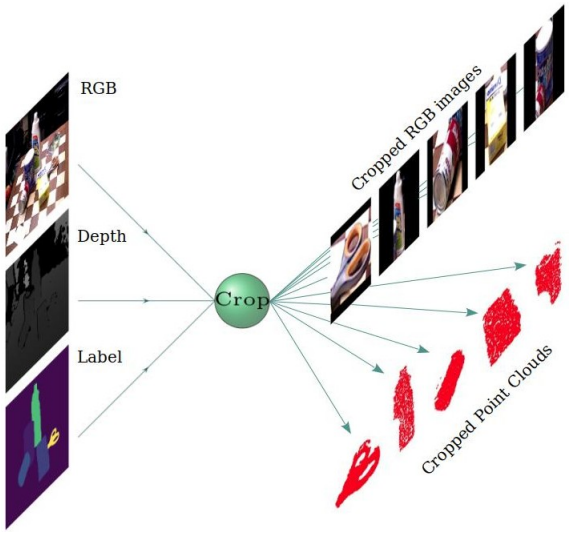

Figure 4.3: Color image crop and point cloud masking illustration.

In [3], it is suggested that each crop is fed into the DenseFusion pipeline one at a time. However, in this implementation, the image crops are stacked up in batches of size $N_{objects}$ (number of object instances in the image). The same is done with the point clouds. This is done this way for two reasons: at first, this method is more efficient because we are able to exploit the GPU resources better. Since the main advantage of a GPU is the ability to perform parallel processing, we want to feed it with as many tasks as possible at a time, for instance, to ask it to perform convolutions on multiple images in just one step. This will not only reduce the training time but also the overall inference time of the model. The second reason is the belief that image crops belonging to the same image hold relevant information of the scene itself and how the objects are displaced relative to each other and the scene. Hence we might be able to not only accelerate the training process but also achieve better accuracy.

The RGB image crops are performed by cutting it using the outermost pixels of the segmented masks as borders. Since we want to create batches in this implementation, the image crops are then resized to a fixed resolution of $192 * 128$, keeping their original aspect ratio and filling the borders with black. On the other hand, the point cloud masking is performed by mapping the labeled and depth images and directly deprojecting a sample of pixels belonging to an object using the camera intrinsics. If the number of pixels belonging to an object is less than the sample size, then the point clouds are filled with points at the origin, so these match in size, allowing to create batches.

- RGB feature extraction block (CNN): Similar to the semantic segmentation block, this is a convolutional neural network arranged in an encoder-decoder configuration. The input to this branch is the batch of color image crops, and the outputs are feature maps with the same resolution and channel size 128. This block is composed of a ResNet-18 followed by four up-sampling layers as the decoder.

  A ResNet (Residual Network) is a state-of-the-art architecture that was originally introduced in [4]. It is one of the most used deep learning models nowadays and emerged as a solution to a curious problem that caught its authors' attention. Even after using batch normalization layers, they noticed that using more layers in a certain CNN was performing worse than the same network but with fewer layers. They also found that this underwhelming performance was not being caused by overfitting, so this had to be a training issue. This phenomenon is counterintuitive because we always expect a network to perform at least as good as its smaller version. Let us consider, for instance, a trained network with 20 layers, if we add 30 more layers that do nothing at all or, in other words, perform an identity operation, we should expect the network to have identical performance. However, SGD [1] seems to struggle in finding this solution in large networks. Therefore, the authors' propose a different way of adding these extra 30 layers, which is to feed-forward the output of each convolutional layer to the next layer's output. In fact, the paper proposes a variant of this approach, which is to skip over every second convolution as demonstrated in Figure 4.4.

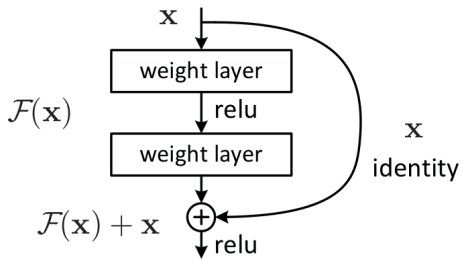

Figure 4.4: Skip over connection on a ResNet. [4]

If each layer is composed of a convolutional layer plus batch normalization, and if the parameter gamma [2] of the batchnorm is initialized to zero, then the identity mapping is achieved while leaving trainable parameters on the convolutional layers. In this manner, the authors showed that a larger network could indeed achieve results at least as good as a smaller version of it. Now, the network's task is to learn residuals, in other words, slight differences between the inputs and the outputs, which is arguably an easier task. That is why this architecture outperformed the state-of-the-art when it was released in 2015, and it is still in the lead.

[1] Stochastic Gradient Descent

[2] Gamma is the slope of the linear function that defines the batch normalization layer.

The implemented ResNet on this research is "eighteen" layers deep (without bottleneck layers) and uses some changes from [90]. It starts with three plain convolutional layers followed by max-pooling, which is called the *Stem* of the network. The underlying principle behind this initial net is that most of the work in a CNN is done in the early layers, so these should be kept fast and simple. The first layer has a stride of 2, reducing the image size to $\frac{1}{2}$ of its original size. The other trick is the usage of *Bottleneck Layers*, where instead of using two convolutional layers with kernel size $3 * 3$ on the residual block, it uses two layers with kernel size $1 * 1$ and one layer with kernel size $3 * 3$ in the middle as shown in Figure 4.5, making a total of 26 hidden layers instead of 18. This is proven to be faster to compute in [90], thus improving inference time and allowing to use a larger number of filters on the layers. In Figure 4.6, the RGB feature extraction network is illustrated.

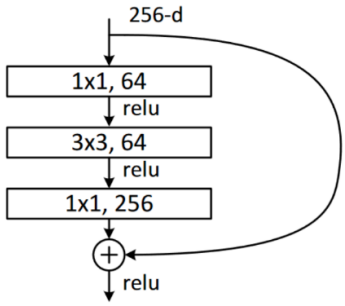

Figure 4.5: Bottleneck Skip over connection on a ResNet. [5]

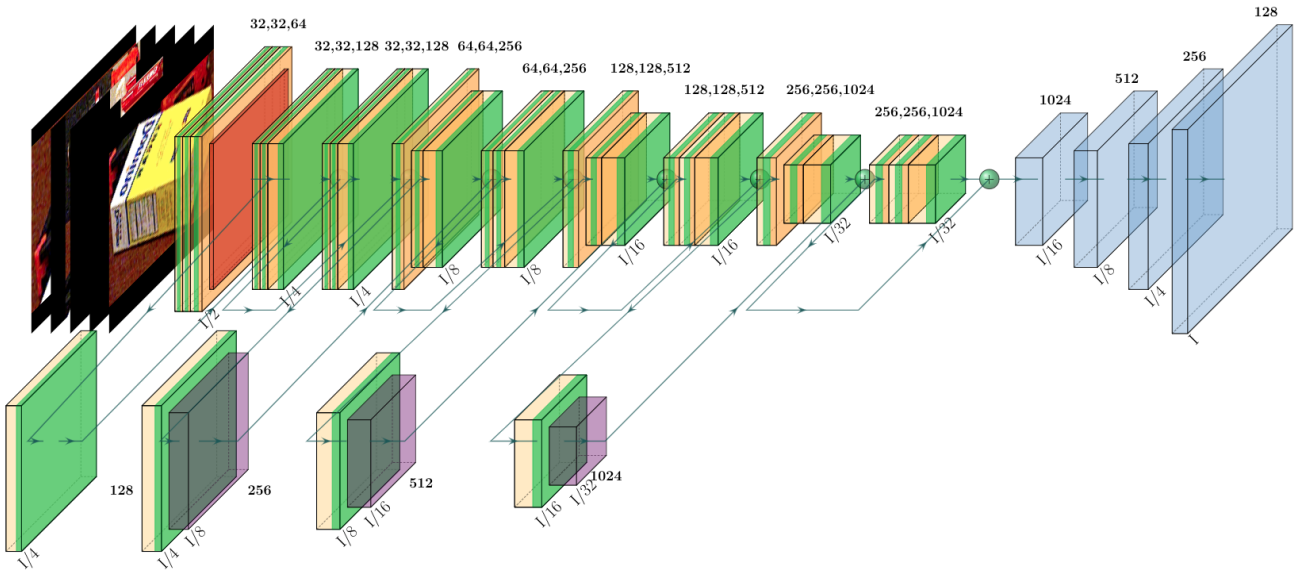

Figure 4.6: RGB Feature Extraction network. *Yellow boxes are convolutions. Blue boxes are deconvolutions. Red boxes are max pooling. Purple boxes are average pooling. Green bands represent batch normalization. Dark yellow bands represent ReLU activation. After each sum, there is a ReLU activation that is not represented in the illustration.*

- Point cloud feature extraction block (PointNet): In DenseFusion, the point cloud features are extracted with a variation of another well-known model, PointNet [91].

  As opposed to the previously mentioned models, this is not a convolutional neural network. Instead, PointNet uses MLP's [3] on its blocks. It directly takes a raw point cloud and computes a global feature vector describing the whole point cloud. This global feature can then be used as the input for another MLP to perform tasks such as object classification, point cloud segmentation, and, in this case, for 6DoF object pose estimation.

  PointNet established a remarkable improvement in the point cloud processing field, as it pioneered the usage of symmetric pooling functions to compute global features, allowing it to achieve permutation invariance. The network computes per-point features and, in the end, it uses a max-pooling reduction function to compute a global feature. Hence the order in which the points are presented to the network does not matter. The authors of [91] also show how to achieve transformation invariance by using blocks that learn to correct the point cloud to its canonical form.

  In DenseFusion's paper, a variant of this model is proposed, where instead of using max-pooling as the reduction function, it uses average pooling that, instead of computing a global feature vector, it computes per-point features only, and the reduction function is used to reduce the dimension of the feature vectors instead of the point cloud dimension. The original net takes a point cloud of shape $N_{points} * 3$ as input, where each row of the point cloud matrix is a vector holding the coordinates (x, y, z) of a point, and outputs a global vector of size $1 * 1024$. On the other hand, DenseFusion's variant outputs a matrix of size $N_{points} * 128$. Besides that, the variant does not use the transformation correction blocks. Otherwise, we would be losing important information relating to the object's pose.

  In Figure 4.7, the implemented point cloud feature extraction network is illustrated.

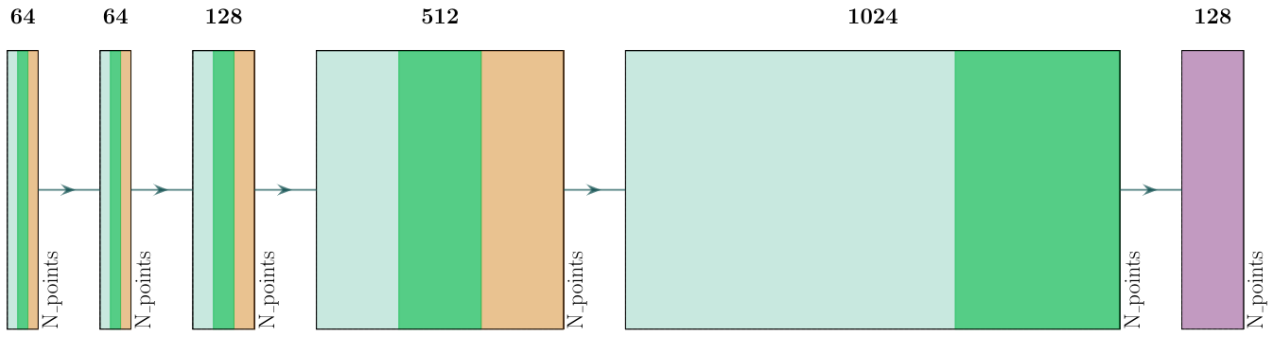

Figure 4.7: Point cloud Feature Extraction network. *Light green rectangles are MLP's. The purple rectangle is average pooling. Green bands represent batch normalization. Dark yellow bands represent ReLU activation.*

[3] Multi-Layer Perceptrons

- Pixel-wise DenseFusion block: The outputs of the RGB and point cloud feature extraction blocks sizes are $N_{objects} * N_{channels} * H * W$ and $N_{objects} * N_{points} * N_{channels}$ ($N_{channels} = 128, H = 128, W = 192$) respectively. Then, in the Pixel-wise DenseFusion block, color and geometric features are concatenated into a feature map of dimension $256 * N_{features}$. The concatenation of the two feature maps is done by mapping the color features with the corresponding geometric features. Since the color and geometric feature maps have different sizes, we have to sample them so these match in size. The size of this sample is the parameter $N_{features}$ that, in this implementation, is chosen to be 1536. In order to save computational effort, the sampling of the point clouds is done before the PointNet, in the crop function. Once the two feature maps are mapped and concatenated, the resulting matrix is fed to an MLP composed of three linear layers, followed by an average pooling reduction function as illustrated in Figure 4.8. This network will output a global feature vector of size 128 describing each of the objects. In the end, this global feature is concatenated with each of the extracted and already concatenated features from the previous block, resulting in a feature map of dimension $N_{objects} * N_{features} * 384$. This is the so-called DenseFusion procedure.

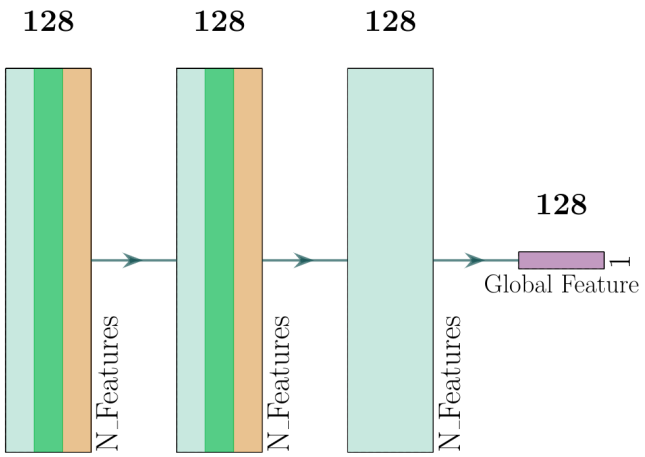

Figure 4.8: Pixel-wise DenseFusion network. *Light green rectangles are MLP's. The purple rectangle is average pooling. Green bands represent bacth normalization. Dark yellow bands represent ReLU activation.*

This is the main novelty introduced in [3], where instead of using only the object's global feature to estimate the object's pose, the model uses this vector solely to enrich the pixel-wise features, giving them global context and letting each of them vote for a pose. Since the segmentation is not perfect, and it is impossible to avoid occlusions, both image crops and masked point clouds will always inevitably contain information belonging to other objects or the background. Thus, it would be naive to rely on global representations of the crops strictly.

- 6DoF pose estimation block: This block is composed by another MLP. In this case, it has four linear layers that take the $N_{objects} * N_{features} * 384$ feature map and outputs a vector of size $N_{objects} * N_{features} * 13$. This vector's elements are the 9 values of a rotation matrix, 3 values of a translation vector, and one more value that represents the confidence score of the estimated pose. This last value enables the network to learn in a self-supervised manner, as it can self-evaluate the quality of its predictions. In the end, the pose with the highest score is chosen as the final prediction.

  The 6DoF pose estimation network is illustrated in Figure 4.9.

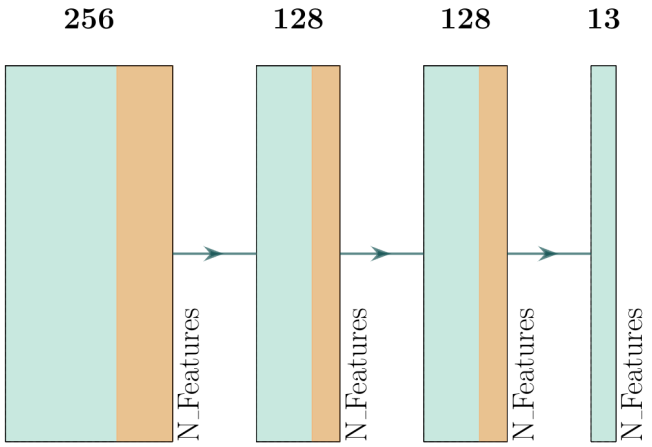

Figure 4.9: 6DoF pose estimation network. *Light green rectangles are MLP's. Dark yellow bands represent ReLU activation.*

4.2.2 Proposed Change to the Original Architecture

In addition to the improvements on the crop function and RGB feature extraction block, namely the use of batches and bottleneck layers, respectively, there is another proposed tweak to the original architecture, described and compared in an independent section, as this one is more complex.

One of the main arguments behind DenseFusion concept is that color and depth images should be treated differently, as these are defined in different domains. However, in the 6DoF pose estimation block, the rotation, translation, and confidence score, which are as well defined in distinct domains (Translation $\in SE(3)$, Rotation $\in SO(3)$, Confidence score $\in \mathbb{R}$) are being computed all together in the same network. Even though this is not problematic, as the weights of the last layer should adapt to the different domains, in this research, there is the belief that it is possible to improve the overall accuracy of the model if we use independent blocks for the two transformations plus the confidence score. Therefore, a different 6DoF pose estimation network is proposed, where this idea is implemented and compared to the original network. In section 4.2.7, the results of the changed DenseFusion are presented, where it is shown that

we can indeed improve the overall accuracy by separating these parameters' estimation.

A problem with this approach is that, if the confidence score network is implemented the same way as the rotation and translation networks, i.e., taking the outputs of pixel-wise DenseFusion block outputs as inputs, it will then lose track of the final pose, making it harder to evaluate if the final pose is good or not. Thus, the inputs to the confidence score network are the outputs of the second last layers of the rotation and translation blocks, evaluating the pose not by its final result but by learning to find patterns on the last feature maps that yield a good or bad result.

In Figure 4.10, the new 6DoF pose estimation block is illustrated.

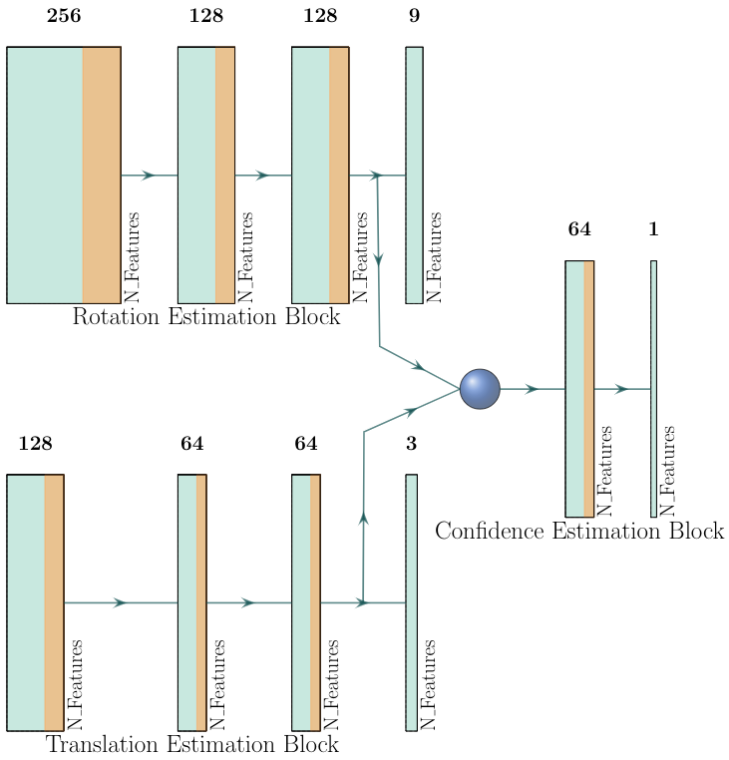

Figure 4.10: Changed 6DoF pose estimation network. *Light green rectangles are MLP's. Dark yellow bands represent ReLU activation. The dark blue ball represents concatenation.*

4.2.3 Evaluation Metrics

- Segmentation network evaluation metric: To evaluate this network, *Mean Intersection over Union* (MIoU) is used.

  Also known as the Jaccard Index, this loss function determines the average of the intersection over the union ratio of the ground-truth labels and predicted labels. It can be seen as a similarity measure, which compares the similarity and diversity of the two sets. It is computed as follows:

$$MIoU = \frac{1}{N_{classes}} \sum_i \frac{n_{ii}}{\sum_j n_{ij} - n_{ii} + \sum_j n_{ji}} \tag{4.1}$$

Where n_{ij} is the $(i,j)^{th}$ parameter of the confusion matrix, i.e., it represents the number of pixels with true class label i while the predicted class is j. Hence, n_{ii} is the number of correctly predicted pixels, $\sum_j n_{ij}$ is the total number of pixels that belong to class i, and $\sum_j n_{ji}$ is the total number of pixels that were labeled as class i.

- DenseFusion's evaluation metrics:
  - Average Distance (ADD(-S)) - Given the 3D model, ground truth and estimated poses of an object, m, $[R|t]$ and $[R'|t']$ respectively. This metric will output a score ω_{ADD} as an accuracy measurement of the estimated pose, the smaller the better. This score is computed as follows:

$$\omega_{ADD} = \frac{1}{M} \sum_i ||(Rx_i + t) - (R'x_i + t')|| \tag{4.2}$$

Where M denotes the total number of sampled points of model m, x_i is i^{th} sampled point of the model, R and R' are the truth and estimated rotation matrices respectively, t and t' are the truth and estimated translation vectors respectively.

If the model is symmetric, i.e., it has ambiguous views, the equation above is adapted so it measures the average distance of the closest points. This variant of the ADD metric is denoted as ADD-S and is computed as follows:

$$\omega_{ADD-S} = \frac{1}{M} \sum_i \min_{0<k<M} ||(Rx_i + t) - (R'x_k + t')|| \tag{4.3}$$

Where x_k is the closest point to x_i.

This metric can be used both as a loss function and an accuracy measurement. In the latter case, hypotheses for which the ADD(-S) score is less than a given threshold are accepted as correct. In most cases, this threshold is defined as follows:

$$\omega_{ADD(-S)} \leqslant k_m d \tag{4.4}$$

Where k_m is a chosen coefficient and d is the diameter of the object.

  - Area Under Curve (AUC) - As opposed to the metric presented above, this metric doesn't follow a specific expression and cannot be used by itself. By definition, this metric measures the area under the curve defined by some other function. In the 6DoF pose estimation literature, this metric is used to compute the area under the ADD(-S) curve as a function of the chosen threshold mentioned in inequation 4.4.

4.2.4 Loss Functions

Since the semantic segmentation and DenseFusion networks have different jobs, these need to be trained separately. Accordingly, each of them has its unique loss function.

- Segmentation network loss function: To train this network, *Cross Entropy* loss function is used.

  Since this is a classification network in its basis, and that there is not that much intricacy about measuring if a label is well chosen or not, a generic loss function for classification models is used.

  The outputs of the last layer are not actual labels, these are real numbers. So what we want here is that the network learns to activate the channels belonging to the respective label more than the others. Recall that the number of channels in the last layer is the same as labels. To do so, the cross entropy loss function is used, that firstly computes softmax on the outputs of the network, and then uses the ground truth labels as indexes to get the corresponding channel values and subsequently calculates the errors. Since softmax transforms the outputs of the model into probabilities, all we need is to measure the difference between the indexed predictions and 1. However, it is much easier for the network to distinguish values that variate in between negative and positive infinity than 0 and 1. Therefore, in the loss function, the *Negative Log Likelihood* of the error is also computed, so its interval changes from $[0; 1]$ to $[-\infty; +\infty]$. This is basically taking the logarithm of the error and multiplying by -1 to invert the signal of the loss, because we want to minimize the error and not maximize.

  The *Cross Entropy* loss function can be written as:

$$L_{CE} = -log\Big(softmax(p, t)\Big) = -log\left(\frac{e^{p[t]}}{\sum_i e^{p[i]}}\right) \tag{4.5}$$

  Where p and t are predictions and targets respectively.

- DenseFusion's loss function: It follows the same expression as the ADD(-S) metric but, as we want the network to learn to balance the confidence among the per dense-pixel predictions, the confidence score is added to the expression as follows:

$$L_{ADD(-S)} = \frac{1}{N_{Features}} \sum_i \Big((w_{ADD(-S)})_i^t c_i - \omega log(c_i)\Big) \tag{4.6}$$

  Where t is the target pose, c_i is the i^{th} calculated confidence score and ω is a balancing hyperparameter. As suggested in DenseFusion's paper, ω is set to 0.01.

4.2.5 Software and Hardware

The models were implement in Python, using the PyTorch [92] framework and some functions from the Fastai [93] library. Training and evaluation was done in a Desktop with an AMD Ryzen 7 3800X CPU, 16GB of RAM, a Samsung 970evo plus SSD, and a NVIDIA GeForce RTX2080ti GPU.

4.2.6 Training and Evaluation

So far in this Chapter, we already have all the tools to train and test a machine learning model geared towards pose estimation. Nevertheless, there is still a crucial topic to address, which is how these tools are used to train and test the model. Not just DenseFusion, but also the semantic segmentation network that requires its own training stage.

Thereupon, this section is dedicated to demonstrate how these steps are performed.

- Dataset preparation: The YCB-Video dataset is split into 80 videos for training plus the 80000 synthetic images, and 2949 key frames chosen from the remaining 12 videos for testing. The frame resolution is $640 * 480$. In DenseFusion case, color images and point clouds are firstly normalized and then the crop function takes place, creating batches of image crops and masked point clouds. On the other hand, to train the segmentation network, only normalized RGB images are used and these are stacked in batches of size 8. Each batch is fed to the network at a time and in different order in each training epoch. Normalization of the data facilitates training and shuffling helps the optimizer avoiding local minima.

- Optimizer: One of the main challenges of neural network optimization is to avoid local minima. SGD updates the net weights by subtracting them with the product of learning rate times the weights' gradient. This optimizer is highly prone to fall into local minima because as some weights gradient reaches zero, the algorithm will stop updating that weight even if it is not global minima. Accordingly, two solutions can be used: Momentum and Adaptive learning rate. The first one adds the moving average of the n previous weight steps to the SGD equation, so these are updated not just based on the gradients but also on how it was being updated before, thus giving "momentum" to this value and hopefully helping it skip sub-optimal solutions. The second one adapts the learning rate to each weight individually so that weights that had been very close to zero in the previous updates receive a slightly higher learning rate, giving them a "push" so these can jump out sub-optimal solutions.

  An optimizer that combines these two solutions is *Adam* [94]. Therefore, this is the tool used to

optimize both DenseFusion and the semantic segmentation network.

- Training policy: This is where the methodologies used for DenseFusion and the semantic segmentation network differ the most, so these will be explained separately:

  - DenseFusion: Considering the very large size of the YCB-Dataset $(410GB)$, each training epoch took about 10 hours to compute. Thus, it would have been very inefficient to let the optimizer run several epochs without taking care of the learning rate or even storing the net weights, risking losing it all if the machine crashed or other critical events happened. Because of that, a cached epoch training policy was adopted. Where the model weights were stored after each epoch and, the learning rate was decreased by half whenever the validation accuracy dropped. If that happened, the second last epoch was used to continue training, throwing off the last epoch, where the validation accuracy got worse.

    The initial learning rate was set to 10^{-4} and decreased to 10^{-6} in a total of 32 epochs. Taking about two weeks to train. The training process was stopped when the validation accuracy stopped increasing. For the sake of comparability, both original and tweaked DenseFusion's versions were trained in the exact same manner.

  - Semantic segmentation network: Since this model is smaller and batches of 8 images were used, it was much faster to train. So the problems found while training DenseFusion were not relevant here. Nevertheless, to improve training time, the model was firstly trained for 24 epochs with the same images but at half of the original resolution $320 * 240$, taking 20 hours to train, and the learning rate was progressively reduced from 10^{-3} to 10^{-6}. Then, the model was trained for more 4 epochs at full resolution by reducing the learning rate by half at each epoch.

4.2.7 Results

In this section, the results of the implemented models are presented, joined by inference time results.

✣ Accuracy

In table 4.1, the accuracy results of both original and tweaked DenseFusion implementations are presented. As we can see, the proposed tweak does improve the original architecture on the ADD(-S) and AUC metrics. It is worth highlighting that the results in this table were obtained from two implementations done in the practical work of this thesis, not from the original DenseFusion paper. Thus, a fair comparison of the models can be done, as the only difference in the code is the mentioned tweak.

The segmentation network obtained $95,5\%$ accuracy on the MIoU metric.

	Original DenseFusion		Tweaked DenseFusion	
	AUC	<2cm	AUC	<2cm
master chef can	95,9	100,0	96,4	100,0
cracker box	92,0	99,1	95,6	99,8
sugar box	95,8	100,0	95,9	100,0
tomato soup can	93,8	97,5	95,0	98,5
mustard bottle	95,5	99,8	96,0	100,0
tuna fish can	95,3	100,0	95,2	100,0
pudding box	92,2	97,8	94,3	99,7
gelatin box	95,1	99,9	95,8	100,0
potted meat can	91,4	93,2	92,4	95,2
banana	91,0	93,8	91,9	94,6
pitcher base	93,1	98,7	94,3	99,9
bleach cleanser	92,7	98,2	93,9	99,6
bowl*	84,9	75,1	90,6	80,1
mug	96,3	99,9	96,7	100,0
power drill	91,8	96,9	93,0	98,0
wood block*	86,1	92,8	89,9	96,5
scissors	96,0	100,0	96,3	100,0
large marker	94,1	98,2	95,5	99,7
large clamp*	68,6	76,8	73,5	81,1
extra large clamp*	67,9	69,9	72,1	73,0
foam brick*	94,5	99,6	94,8	100,0
AVG	90,7	94,6	92,3	96,0

Table 4.1: Implemented 6D pose estimator's performances on the YCB-Video dataset in the ADD(-S) metric for $\omega_{ADD} < 2cm$, and also AUC in terms of ADD(-S) metric for $0 < \omega_{ADD} < 10cm$. Objects with a "*" next to them are symmetric.

✣ Inference Time

Table 4.2 shows the inference time of the implement models or, in other words, computation time. To present a more solid insight into how the models perform in terms of computation time, in these measurements, instead of only using the extracted key-frames, all frames from the remaining 12 videos are used. In this manner, we can better demonstrate if the models are proper for real-time application, furthermore, what is the impact of the proposed tweak in terms of computation time. It is worth mentioning that this tests are done on a dataset stored on the machine SSD, and its performance has a strong impact on the inference time of the models, because all the frames are being loaded from it. Moreover, the 6DoF pose estimator is not composed solely by the DenseFusion network. Therefore, all the steps required to estimate the objects' pose are discriminated in the following table.

	Mean	STD	VAR	MAX	MIN
Calculate poses (w/ tweaked version)	0,0513	0,0021	4,25E-06	0,0797	0,0460
- Load RGB + PCL	0,0194	0,0015	2,35E-06	0,0321	0,0167
- Segmentation Network	0,0153	0,0004	1,56E-07	0,0233	0,0150
- Crop RGB + PCL	0,0084	0,0014	1,87E-06	0,0167	0,0047
- Original DenseFusion	0,0069	0,0004	1,97E-06	0,0100	0,0058
- Tweaked DenseFusion	0,0072	0,0004	1,97E-07	0,0120	0,0064

Table 4.2: Implemented models' inference times on GPU. All values are in seconds.

As we can see from the results shown above, both models can perform in a real-time scenario, at an average of 20 frames per second. Moreover, it is also shown that the proposed tweak has minimal impact on the inference time. The very satisfactory results shown here can be justified not only by DenseFusion's design and the aforementioned improvements, but also because a rather powerful GPU is being used. For instance, if we would compute the models only on CPU (which is also a relatively powerful one), the inference times would be certainly underwhelming, as shown in table 4.3.

	Mean	STD	VAR	MAX	MIN
Calculate poses (w/ tweaked version)	1,0160	0,1010	1,02E-02	1,3496	0,9463
- Load RGB + PCL	0,0194	0,0015	2,35E-06	0,0321	0,0167
- Segmentation Network	0,6268	0,0465	2,20E-03	0,8032	0,5648
- Crop RGB + PCL	0,0089	0,0008	6,47E-07	0,0146	0,0080
- Original DenseFusion	0,3247	0,0277	1,97E-06	0,4363	0,2764
- Tweaked DenseFusion	0,3598	0,0278	8,00E-04	0,4591	0,3112

Table 4.3: Implemented models' inference times on CPU. All values are in seconds.

Chapter 5

Machine learning based approaches vs 3D Point cloud registration

5.1 Introduction

TEASER++ [1] is a very recent 3D point cloud registration algorithm that showed excellent results for point cloud registration in the presence of large amounts of outlier correspondences while (as the authors state) being able to perform these calculations in real-time.

Considering its performance, one of the main motivations to start this thesis was the belief that this algorithm could have been used to aid a 6DoF pose estimator in terms of inference time. The underlying reasoning of this idea is simple; machine learning based models often suffer from poor inference times and, because of that, one of the main challenges related to the 6DoF object pose estimation field is to develop an architecture that can achieve satisfactory accuracy while being able to perform in real-time. On the other hand, a 3D point cloud registration algorithm such as TEASER++ can be much less computationally intensive, thus being faster to compute. However, such algorithms cannot estimate absolute poses. These algorithms estimate a relative pose between two point clouds. So, how can we fuse such techniques? One possibility is to use a 6DoF pose estimator to estimate the initial pose of an object, and from there, we can use point cloud registration to track the pose by estimating the relative poses of the objects in subsequent frames.

The concept of tracking objects with point cloud registration is not new. Some works, such as [95], [19], and [41] already exploited this idea. However, because TEASER++ is very recent, there is no relevant reference in the literature where this algorithm has been used as a 6DoF pose tracker. Therefore, this

idea will be exploited in this Chapter. Furthermore, a comprehensive discussion on the topic will also be presented.

5.2 Methodology

5.2.1 TEASER++ [1]

TEASER stands for Truncated least-squares Estimation and Semidefinite Relaxation. One of the biggest challenges related to 3D point cloud registration is to find point-pair correspondences, i.e., having two point clouds representing the same object, we want to match corresponding objects. For instance, if we are dealing with CAD models, the point clouds are well defined, and one can expect to find all correspondences. However, this is not the case when the point clouds are obtained with a RGB-D sensor (as an example), where the data is noisy and contains large amounts of points that do not even belong to the object. For that reason, the authors of TEASER purpose truncated least-squares estimation, which can discard outlier point-pair correspondences from the least-squares problem. This is done by giving a new parameter to the least-squares equation that establishes a residual limit. By doing so, the method can discard matched points with large residuals. The method also employs semidefinite relaxation in the rotation estimation step.

The truncated least-squares problem can be defined as follows:

- Under the assumption that the noise on the point clouds is unknown but bounded, such that β_i where $||\epsilon_i|| \leq \beta_i$ is the given bound. The truncated Least Squares registration problem is formulated as follows:

$$\sum_{i=1}^{N} \left(\frac{1}{\beta^2{}_i} ||b_i - sRa_i - t||^2, \underline{c}^2 \right) \tag{5.1}$$

  Where N is the size of point sets A and B, a_i and b_i are the ith points of point sets A and B respectively, s is a scale, R is a $3D$ rotation, t is a $3D$ translation, and $\underline{c}^2$ defines the residual threshold.

  Therefore, equation 5.1 computes a least-squares solution of measurements with small residuals $\rightarrow \frac{1}{\beta^2{}_i} ||b_i - sRa_i - t||^2 \leq \underline{c}^2$, while discarding large residuals $\rightarrow \frac{1}{\beta^2{}_i} ||b_i - sRa_i - t||^2 > \underline{c}^2$.

The authors acknowledge that solving least-squares estimation in such a manner is hard to solve compared with previous methods like RANSAC. Hence, a method for decoupling scale, rotation, and translation is

also proposed. By doing so, these transformations can be estimated independently. The key idea is that the problem can be reformulated in order to obtain quantities that are invariant to a subset of these transformations.

Before describing how this is done, let us first repeat equation 2.1 from Chapter 2, as it will be helpful for the following demonstrations.

$$b_i = s^{\circ}R^{\circ}a_i + t_i + o_i + \epsilon_i \tag{5.2}$$

The decoupling of the approximation problem is achieved as follows:

- Translation Invariant Measurements (TIMs): While the absolute poses of the points in a point cloud depend on the translation, their relative poses not. This is easily proven if we write the following equation:

$$b_j - b_i = sR(a_j - a_i) + (t_j - t_i) + (o_j - o_i) + (\epsilon_j - \epsilon_i) \tag{5.3}$$

  Since the relative poses of the points are the same, then the translation cancels out, yielding:

$$b_j - b_i = sR(a_j - a_i) + (o_j - o_i) + (\epsilon_j - \epsilon_i) \tag{5.4}$$

  And allowing to write the Translation Invariant Measurement (TIM) as follows:

$$b_{ij} = sRa_{ij} + o_{ij} + \epsilon_{ij} \tag{5.5}$$

- Translation and Rotation Invariant Measurements (TIMs): The relative locations of pair of points (TIMs) still depend on the rotation, but their distances does not. Hence, another invariant measurement can be created if we compute the norm of the TIMs:

$$||b_{ij}|| = ||sRa_{ij} + o_{ij} + \epsilon_{ij}|| \tag{5.6}$$

  In case of inliers $(o_{ij} = 0)$, and if we note $\beta_i + \beta_j = \delta_{ij}$, it holds:

$$||sRa_{ij}|| - \delta_{ij} \leq ||sRa_{ij} + \epsilon_{ij}|| \leq ||sRa_{ij}|| + \delta_{ij} \tag{5.7}$$

  This way, 5.6 can be rewritten as:

$$||b_{ij}|| = ||sRa_{ij}|| + \tilde{o}_{ij} + \tilde{\epsilon}_{ij} \tag{5.8}$$

Since the norm is rotation invariant and $s > 0$, the TRIMs can be obtained by dividing both sides of 5.8 by $||a_{ij}||$:

$$s_{ij} = s + o^s_{ij} + \epsilon^s_{ij} \tag{5.9}$$

With that said, the structure of the Truncated least-squares Estimation and Semidefinite relaxation can be summarized as follows:

I TRIMS are obtained and used to estimate the scale $\hat{s}$.

II TIMS are obtained and combined with ŝ to estimate the rotation $\hat{R}$

III $\hat{s}$ and $\hat{R}$ are used to estimate $\hat{t}$ from the point matches in the original truncated least-squares problem.

Both the steps mentioned above are solved using robust techniques by maximizing a consensus set.

5.2.2 Pose Tracking Using 3D Point Cloud Registration

The main goal of this experiment is to evaluate if a 6DoF pose estimator's inference time can be improved by fusing it with a 3D point cloud registration algorithm.

The adopted approach to perform this experiment is to pick one of the implemented DenseFusion's versions from Chapter 4, use it as an initial pose estimator, and TEASER++ as a pose tracker. Since the point clouds of each object in the scene are available, we can use them to compute the relative pose of each object in subsequent frames and then use them together with the initial poses to calculate the absolute poses.

This approach is then split into two main strategies: 1) To track each object individually using the correspondent masked point clouds. 2) To use the whole point cloud of each frame and compute only one relative pose. If we consider that the objects do not move during frames, they should move relative to the camera equally, so we only need to track the camera movement to infer the objects' absolute poses. Indeed, this constitutes a disadvantage compared with the first strategy: the objects can not move independently; otherwise, we lose track of their poses.

Besides these two strategies, there are two approaches in which the tracker will be implemented. The first approach is LUF (Last Updated Frame) which calculates the relative poses of the objects between the current frame and the last updated frame. The other approach is SF (Subsequent Frame), which calculates the relative pose using subsequent frames. The latter approach is expected to perform worse as the error will be incremented in each new frame, in other words, the error is carried out.

The desired way of implementing this tracker with the LUF approach would be to update the poses only when the estimated poses deviate from a given threshold. However, this approach would require measuring the error in real-time, requiring more computational effort and negatively impacting the inference time. Hence, the average number of frames for which the tracker starts to deviate from a given threshold will be measured *a priori*, and then this number will be used to update the poses periodically.

5.2.3 Point-pair Correspondence Search

One of the main challenges of point cloud registration is to find a good point-pair correspondence search algorithm. As it was said already, if we are dealing with computationally generated point clouds, this task is easy because the points follow well-defined mathematical models, so we can expect to find all correspondences with minimal effort. However, this is not the case under study. We want to find correspondences in very noisy and unregular data sourced by some RGB-D sensor. Besides that, the measured depth points possess a rigid relative position with the sensor, which means that the points move along with the camera. Therefore, point clouds obtained from different frames do not even represent the same points in space.

The last difficulty mentioned above is vital to understand the algorithms that will be addressed in this section. The main task of these methods is to first find features in the point clouds, and then, the points can be matched by comparing their features.

In [1], the authors demonstrate how TEASER++ can be used without finding matches first. Since the truncated least-squares estimation is cable of rejecting outlier correspondences, one can feed TEASER++ with all possible matches. This way, outlier matches would be rejected, leaving only inliers.

The evident problem with this approach is that we are feeding the algorithm a much larger number of correspondences. More specifically, if each point cloud has N points, then the number of matches would be N^2. This is much more computationally demanding, thus much slower to compute and improper for a real-time application.

Considering all these facts, four methods were tested, and for the sake of organization, this methods will

be described in the following list:

- Strategy 1) with SIFT: This method consists of using SIFT [1] [96] to detect features on the RGB image crops containing each object. By detecting features in the color images, we can then match them and compute the points using the pixel coordinates and the known camera intrinsics, thus constructing the input point clouds for TEASER++. A similar option was to use SURF [2] [97] instead, but this algorithm was not available in the OpenCV library as it is still under its patent ownership.

- Strategy 1) with FPFH: While with the previous approach, the color images were being used to provide features, this method directly searches for features in the point clouds. To do so, FPFH [3] [42] is used. By directly computing key points from the point clouds it is expected, at least, to achieve better inference times.

- Strategy 1) with 6-PACK: This approach also computes 3D key points directly from the point clouds but, this time, a more robust algorithm was used. In this try, a machine learning based feature detector and matcher algorithm is considered: 6-PACK [4] [19].

  In [19], a similar idea to that addressed in this Chapter is exploited, i.e., to estimate the initial pose of the objects using an independent 6DoF pose estimator and then track them using a point cloud registration algorithm, in their case, the authors use ICP [5]. Even though this work encapsulates the 6DoF pose estimation and tracking as a whole, its main contribution is the proposed key point detection approach. This model uses a dissected DenseFusion network to compute features from the RGB image and depth maps. Then, they use these features to compute key points using an attention mechanism over a grid of anchor points generated around the predicted pose of the object. Each anchor summarizes the points around them. By doing so, this model is able to find a coarse centroid of the object, which will guide the following search for key points. Another network then learns the key points in an unsupervised manner.

- Strategy 2) with SIFT: This method consists in instead of tracking each object's relative poses individually, it tracks the camera pose itself. To do so, only the full RGB-D, a not-so-robust key point detector, and TEASER++ are needed. As in this case, more information is available, then it is more likely to find good key point correspondences and a better-defined transformation between point clouds from subsequent frames. In this try, DenseFusion is used to compute the initial absolute

[1] Scale Invariant Feature Transform
[2] Speed Up Robust Features
[3] Fast Point Feature Histograms
[4] 6DoF-Pose Anchor-based Category-level Keypoint
[5] Iterative Closest Point

pose of the objects, SIFT for key point detection and matching, and TEASER++ to compute relative poses.

5.2.4 Accuracy and Inference Time Measurements

Considering that the error on the initial absolute pose estimate is carried out with the following poses by the tracker, the tracker's accuracy in terms of deviation from the ground truth poses will be highly correlated with the initial pose accuracy. Hence, the tracker's accuracy is measured as the number of frames for which the tracker starts to deviate from an initial ground-truth pose by a given threshold. By measuring the tracker's accuracy this way, it is possible to infer the tracker's performance itself and independently from the initial pose estimate's quality because the ground truth poses are being used instead of DenseFusion's estimates. Moreover, this measurement will be taken when the tracker fails to achieve the threshold in three consecutive frames, as it may fail one frame but succeed on the following one. Nevertheless, if it fails in three consecutive frames, it should be considered that the tracker is lost.

The inference time measurements are performed in the same manner as it was done in Chapter 4. The ADD(-S) metric will be used to compute the distance of the tracked pose from the initial ground-truth pose, and the threshold is set to $2cm$.

5.2.5 Software and Hardware

The models were implemented in Python, using the PyTorch [92] framework and some functions from the Fastai [93] library. SIFT is provided by the OpenCV [98] library. TEASER++ implementation in provided in [99]. The FPFH algorithm implementation was obtained in [100]. All the experiments were performed in a Desktop with an AMD Ryzen 7 3800X CPU, 16GB of RAM, a Samsung 970evo plus SSD, and a NVIDIA GeForce RTX2080ti GPU.

5.3 Results

- Strategy 1) with SIFT: Unfortunately, this method yielded very unsatisfactory accuracy, and for that reason, the experiments with this approach were terminated.
- Strategy 1) with FPFH: As it was already said, with this method, it was expected to achieve better inference times. However, the opposite happened, and poor inference times were obtained in favor

of residual accuracy improvement. The reason behind this might be the fact that the FPFH algorithm is iterative and tries to find key point matches by computing normals of small neighborhoods on the point clouds. At first, the K nearest neighbors are found for each point in each point cloud, then the normals are computed by fitting a plane to each neighborhood using singular value decomposition. The normals are then used as features so that the correspondences can be calculated by matching points with identical normals. Therefore, this algorithm takes $K * N$ time to compute, where K is the size of the neighborhood and N is the size of the point clouds. This means that the algorithm will be as slow as large are the point clouds or neighborhoods.

Clearly, a solution to this problem would be to reduce K or N but, reducing the size of the neighborhood negatively impacts accuracy, and sampling the point clouds without knowing the correspondences *a priori* would be a naive approach, as we risk to lose a significant amount of key point correspondence candidates if not all. Having said that, this method was also discarded.

- Strategy 1) with 6-PACK: Unfortunately, this solution also fails. As explained above, 6-PACK still needs DenseFusion to compute the features and, before even going through the implementation of this algorithm, some tests on inference time were performed to evaluate the viability of this approach. That is, to understand if a hypothetic inference time improvement can be achieved. Therefore, we can recall to table 4.2 from Chapter 4, where we can see that DenseFusion itself only takes 10% of the total inference time, where the pose estimation module only takes $1ms$. Since 6-PACK still needs all the steps but the pose estimation module, it would be only possible to achieve an inference time improvement of less than 1.25% in the best case scenario. Furthermore, this residual improvement would be very unlikely to be achieved due to the model's increased complexity and computational effort. Therefore, no further development was performed on this solution.

- Strategy 2) with SIFT: The results of this experiment were far better than the previous approaches. Even though the assumption of rigid objects is needed for this approach to work, the accuracy was very satisfactory, and the relative poses of the objects were now possible to be computed using this idea.

Having found a working solution to this problem, using SIFT to compute correspondences in the full RGB image and using the known camera intrinsics to compute the points, the new architecture could be built. Thus, the results on the following sections were obtained with this method. Based on empirical evaluation, it was chosen to use 15 correspondences for registration. The pose tracker was tested on the complete 12 videos left for evaluation of the YCB-Video dataset.

5.3.1 Accuracy

Table 5.1 shows the accuracy results in terms of number of frames for which the tracker looses track of the objects' poses. The measurements were performed in both approaches, LUF and SF. Here, we can see that the first approach is indeed better than the second one.

	Mean	STD	VAR	MAX	MIN
LUF	31,81	24,68	608,85	191	1
SF	8,03	3,98	15,86	30	0

Table 5.1: Pose tracker accuracy in terms of the number of frames before it deviates too much from a given threshold ($2cm$).

Being able to track the objects' poses at an average of 31 subsequent frames, we can consider that the LUF approach's performance is rather satisfactory and can be helpful in a practical scenario.

5.3.2 Inference Time

Following the results shown in table 5.1, and considering that it is unfeasible to update frames dynamically, the inference time measurements were obtained with the LUF approach by updating the poses with DenseFusion in every 30 frames. These results are presented in table 5.2.

	Mean	STD	VAR	MAX	MIN	PU
Total (with pose updating)	0,1013	0,0244	0,0006	0,1825	0,0576	CPU/GPU
- Open RGB	0,006	0,0014	2,07E-06	0,0188	0,0053	—
- Detect Features	0,042	0,003	9,04E-06	0,0596	0,0331	CPU
- Match Features	0,0392	0,0206	0,0004	0,1122	0,0062	CPU
- Calculate points	0,0123	0,0014	2,0318	0,021	0,0096	GPU
- TEASER++	0,0005	0,0003	7,01E-08	0,0084	0,0003	CPU

Table 5.2: Pose tracker inference time. All values are in seconds. "PU" stands for Processing Unit. The pose is updated in every 30 frames, using the LUF approach.

Unfortunately, these results show that it is not possible to improve DenseFusion's inference time using this technique. The average total time spent in tracking the objects' poses and periodically updating them with DenseFusion shows that this method can only perform at about 10 frames per second, while DenseFusion alone can estimate absolute poses in every frame at a rate of $20FPS$.

Still, it is important to mention that the relative poses were being calculated by the CPU, while DenseFusion is computed on GPU. If DenseFusion were to be computed on CPU as well, the mentioned approach

would be undoubtedly faster, but still not enough for real-time applications. It is also acknowledged in this thesis that this experiment lacks results for relative pose computed on GPU, which is justified by the fact that a third-party TEASER++ implementation was used, and this implementation does not have CUDA support, making it impossible to run on GPU.

5.4 Conclusion

In this experiment, a 6DoF object pose tracker was built, where a novel 3D point cloud registration algorithm is used, looking forward to understanding if this could be quicker at estimating poses in subsequent frames than using DenseFusion alone. It was concluded that, using the techniques above described, it is not possible to improve DenseFusion's inference time. Nevertheless, the conclusion of this study only applies if we compute DenseFusion on a GPU, if we would use a CPU, it would be worth using TEASER++ to aid the pose estimator, even though it is still not proper for real-time applications. Moreover, only four techniques were addressed. Namely, to use FPFH or 6-PACK to compute point-pair correspondences on each object's point cloud or use SIFT to search for correspondences on the object's RGB image crops and also on the complete color images. It is also acknowledged that it might be possible to construct a very fast pose tracker using the same concept of this study but using 3DSmoothNet [41] as a key point detector and matcher. However, the time boundaries of this research made it impossible to try this technique.

Chapter 6

Relative Pose Study

6.1 Introduction

As the 6DoF object pose estimation field developed through the past years, many metrics were proposed to evaluate the models' accuracy [22], [12], [101]. But what about precision? How consistent are these models with their estimations? Even though not all datasets contain complete videos as the YCB-Video dataset does, this is the scenario that a 6DoF object pose estimator will encounter in most practical applications. That is, their task is to estimate objects' poses in subsequent frames of a video. Therefore, if the objects remain in the same position while the video is captured, one should expect their relative poses to remain constant between frames. However, we do not know if this is the case with the estimated poses. In fact, we do not even know if the ground truth labels provided by the dataset respect this rule.

As far as it is known, no metric was proposed to perform this evaluation. Hence, in this Chapter, it will be demonstrated how this can be done, where a precision metric is proposed. This metric has the great advantage of not requiring ground truth poses. Thus it will be used not only to evaluate the implemented DenseFusion precision, but also the quality of the ground-truth poses of the YCB-Video dataset.

Moreover, it will be also shown how this metric can be adapted to be used as a loss function to train machine learning models for 6DoF object pose estimation.

6.2 Methodology

This study aims to infer the 6DoF pose estimator precision in terms of the relative pose of the objects present in the image. Having multiple images of the same scene but from different perspectives, one should expect the absolute poses of the objects to change. However, if the objects remain in the same positions between images, their relative pose with each other should remain constant. Hence, by measuring how these relative poses change in between frames, it will be possible to not only infer the quality of a pose estimation model, but also from the ground truth poses given on a certain dataset.

6.2.1 Proposed Metric for 6DoF Pose Estimator Precision Measurements

Having a set of frames of size N, each containing k objects, with $N > 1$ and $k > 1$, and the respective poses, and if we consider that the camera captures all objects in every frame, an object can be randomly chosen as a reference, which will be denoted as object o_r. Then, the poses of the other objects relative to the reference object are calculated. This is done for all frames and will result in $k - 1$ relative transformations per frame. As said already, we expect these transformations to be equal in all frames under the assumption that the objects did not move during the video. Therefore, all we have to do now is calculate the differences of correspondent relative poses between different frames.

If we write a 3D transformation as:

$$[R|t] = \begin{bmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

We can compute the relative transformation between the reference object o_r and object j in frame n as follows:

$$^{n}[R|t]_{o_r j} = {}^{n}[R|t]_{o_r}^{-1} * {}^{n}[R|t]_{j} \tag{6.2}$$

This is done for all frames and, when all relative 3D transformation matrices ${}^{n}[R|t]_{o_rj}$ are available, the error between these matrices is measured between relative transformations of consequent frames by calculating the product of the inverse of matrix ${}^{(n-1)}[R|t]_{o_rj}$ by matrix ${}^{n}[R|t]_{o_rj}$. Basically, we are calculating the relative transformation between relative transformations and, if these are the same, we expect the result to be an identity matrix of size $4 * 4$. Hence, the error will be the Frobenius norm of the difference between the resulting matrix and an identity matrix, as demonstrated in the following equation:

$$ {}^{n}\epsilon_{[R|t]_{o_rj}} = ||I_4 - {}^{(n-1)}[R|t]^{-1}_{o_rj} * {}^{n}[R|t]_{o_rj}||_f \tag{6.3} $$

Where ${}^{n}\epsilon_{[R|t]_{o_rj}}$ is the j^{th} element of error vector ${}^{n}\epsilon_{[R|t]}$ (vector containing all $k-1$ transformation errors of consequent frames) between frames n and $n-1$. This will yield N-1 error vectors of size $k-1$ that, in the end, are averaged into a single precision score. This score will be denoted as $\tau_{[R|t]}$ and can be seen as how much the poses "jitter" between frames. The smaller it is, the better.

6.2.2 The Proposed Metric as a Loss Function

In the context of this thesis, the proposed metric was initially developed as a precision measurement tool. However, the metric can easily be adapted to measure the model's accuracy and consequently as a loss function.

Up until now, we were interested in evaluating the similarity between correspondent relative poses of subsequent frames because we expect them to be the same, and this is the exact same thing we look for when training a machine learning model, we want the estimated, and the target poses to be the same. Therefore, if we pick equation 6.5, and substitute the relative poses by an estimated and a target pose, we can infer their similarity. This function can then be back-propagated and hence used to update the weights of a neural network.

Having that said, the proposed metric can now be rewritten as:

$$ L = ||I_4 - [R|t]^{-1}_{target} * [R|t]_{estimated}||_f \tag{6.4} $$

For DenseFusion, the equation can be written as follows. Where the confidence score is also added so the model can learn in a self-supervised fashion:

$$L = \frac{1}{N_{Features}} \sum_{i} (c_i * ||I_4 - {}^{i}[R|t]^{-1}_{target} * {}^{i}[R|t]_{estimated}||_f - \omega log(c_i)) \tag{6.5}$$

When compared with the ADD(-S) based loss function, a great advantage of this metric is that it does not require the objects' 3D models. For that reason, the proposed loss function can be less computationally intensive, thus improving training time. Still, the fact that this metric does not account for symmetric objects constitutes a significant trade-off.

6.2.3 Targets for Precision Measurements

In this thesis, only the implemented DenseFusion's estimated poses and the ground truth poses of the YCB-Video dataset will be measured. However, this metric can be used for any dataset or 6DoF pose estimator, as long as the captured frames correspond to the same scene with multiple object instances that did not move while the images are obtained.

6.3 Results

The results shown here were obtained on the complete videos left for evaluation of the YCB-Video dataset, totaling 20738 frames.

	YCB-Video	**DenseFusion Original**	**DenseFusion Tweak**
$\tau_{[R\|t]}$	0.0011	0.0887	0.0776

Table 6.1: Relative pose study results (precision measurements).

As expected, the YCB-Video Dataset labels are the most precise poses. This results also show that the proposed tweak to the original DenseFusion architecture can achieve better precision on the estimated poses. These results can be seen as how much jitter there is on the estimated poses between consequent frames. Visually, if we fit the 3D models to the RGB images, a low score will result in a smoother representation, while with high scores, the models look like they vibrate during the video. So, this score can be seen as how much vibration it is on the projected 3D models. Figures 6.1, 6.2, and 6.3 illustrate this.

Figure 6.1: YCB-Video ground-truth poses representation.

Figure 6.2: Original DenseFusion estimated poses representation.

Figure 6.3: Tweaked DenseFusion estimated poses representation.

6.4 Conclusion

In this Chapter, a metric for the precision of 6DoF poses was proposed. It was also shown how this metric could be adapted to be used as a loss function. Considering that the targets for precision measurements are reduced in this thesis, it would be interesting to give continuity to the study by evaluating more datasets and more 6DoF pose estimators to make a more rich comparison. Besides that, in terms of the proposed loss function, the intent here was only to demonstrate how easily the metric could be adapted to do so. Therefore, further studies should be employed to understand the feasibility of this metric as a loss fuction.

Chapter 7

Case-study: How does DenseFusion Generalize to a Real-World Scenario and how does it Perform by Training it with a Limited Dataset Using Transfer Learning

7.1 Introduction

So far in the thesis, the effort was put into covering all the tools needed to perform 6DoF pose estimation of object instances. Starting with an in-depth comparison of three RGB-D sensors, followed by how we can implement and adjust a 6DoF pose estimator. A comprehensive discussion was presented on whether or not it is worth using 3D point cloud tracking as an aid to the 6DoF object pose estimator and, in the end, it was also proposed a metric to evaluate not only the quality of the predicted poses of such models but also the quality of the given ground truth poses of a dataset of choice.

In this chapter the results presented in the previous chapters will be applied to a specific problem.

Note that we still do not know if the implemented models generalize to data not belonging to the YCB-Video dataset, in real-time, and also if the RGB-D sensors addressed in Chapter 3 can be used in this field. Also to be evaluated is the proposed metric, namely if it can lead to results in a real-world scenario. Furthermore, it is also relevant to understand how straightforward it is to select such models and tune them to our practical application. That is to say, how can we create our own dataset and train a model

with it, using transfer learning.

To address these topics, the experiment will be organized as follows: at first, one of the three Intel RealSense RGB-D sensors will be chosen and integrated with the implemented code, such that we can substitute the YCB-Video images with a video captured in real-time. Then, an experimental setup will be built so that we can test the models. The models will be evaluated on the constructed setup in terms of accuracy, precision, and inference time. If the results are unsatisfactory, then a dataset is created, and the model will be trained with it using the pre-trained network weights from Chapter 4. The results showed that this was the case, so it will also be demonstrated how the dataset was obtained. In the end, results from this last step are presented.

7.2 Methodology

7.2.1 Materials

In Chapter 3, an experimental evaluation was performed on three RGB-D sensors from Intel. As we could see there, the L515 sensor outperformed the other two sensors. Hence this will be the chosen camera for the current experiment.

When the experiment was first thought about, the idea was to use the same objects as the ones used in the YCB-Video dataset. However, most of them are just a box, e.g., ”cracker box”, ”sugar box”, ”gelatin box”, and more. Therefore it was decided to handcraft these objects, using cardboard and the prints from the original products available online. Five objects were handcrafted, the ”cracker box”, ”sugar box”, ”gelatin box, ”pudding box”, and ”potted mean can”. Unfortunately, either because of the different reflectiveness of the objects surfaces or due to different lighting, the segmentation network showed difficulty in properly segmenting some of the objects. Hence, only the ”cracker box” and the ”sugar box” were actually used in this experiment.

To avoid wrongly segmented pixels in the images, a white canvas was used to cover the background. A LED projector was used as the light source and, for the ground truth labels of the dataset, a four-by-four chess marker was used.

Figure 7.1: Handcrafted objects. *Cracker box at the back left. Sugar box at the back right. Pudding box at the front left. Potted meat can at the front middle. Gelatin box at the front right*

7.2.2 Software and Hardware

The experiment was performed using Python, the PyTorch [92] framework, some functions from the Fastai [93] library, and the OpenCV library [98]. Training, evaluation, and data acquisition were done in a Desktop with an AMD Ryzen 7 3800X CPU, 16GB of RAM, a Samsung 970evo plus SSD, and a NVIDIA GeForce RTX2080ti GPU. The videos were obtained with the help of the pyrealsense2 library [102] (Intel Realsense SDK 2.0 bindings for Python).

7.2.3 Experimental Setup

To create the background scenario, the white canvas was fixed to a wall and extended on top of a table. The objects are displaced on top of the table, and the LED projector is pointed to the scene. All the experiments were performed with the windows covered up, thus avoiding any unwanted daylight and ensuring that all the images are obtained with the same light conditions.

7.2.4 Dataset Acquisition

With the above setup, we can already start the camera and capture a video, but we also want to create a dataset. Hence, we need to solve this problem. The main challenge that occurs is to find out how to

measure the ground-truth poses. Clearly, we can not use a 6DoF pose estimator for this purpose. We need a more robust tool to measure the objects' poses. One way of solving this is to use the known objects' dimensions and solve the PnP [1] problem. Since in this experiment we only use boxes, this is a good solution because all we need is to detect the corners and edges of these boxes in the RGB images and, if the dimensions of the edges are known, as well as the camera intrinsics, it is straightforward to calculate the transformation that projects a 3D model into the 2D image plane. However, the OpenCV library provides an even simpler solution, which is a function that detects the corners of a chess pattern and calculates its transformation using the size of the chess squares. This way, all we have to do is to place a chessboard in front of each object and calculate its pose. Undoubtedly, the pose will not be as accurate as it should be. With this approach we lose at least the center of mass of the object because the chessboard is leaning on the objects' surface. Moreover, the chessboard is put in front of the objects by hand, and it is very unlikely that the detected corner overlaps the center of the object, where we want it to be. However, this is easily solved if we posteriorly correct the pose manually, by projecting the object's 3D model on the image and adjusting the pose until a proper fit is found.

Let us now describe how the dataset is obtained. At first, the camera is pointed to the objects and kept still. Then, the poses of the objects are obtained one by one. Considering that doing this for every frame is unfeasible, an extra pose is calculated to be used as a reference. This is done by placing the chessboard in a random location inside the image frame and by calculating its pose. If we now calculate the relative poses of the objects with the reference pose, the absolute poses of the following frames can be easily obtained if the chessboard is kept in the same place and inside the image frame while the video is recorded. The reference pose is measured in every frame, so the objects' absolute poses can be later calculated using the known relative poses. Figure 7.2 illustrates this process.

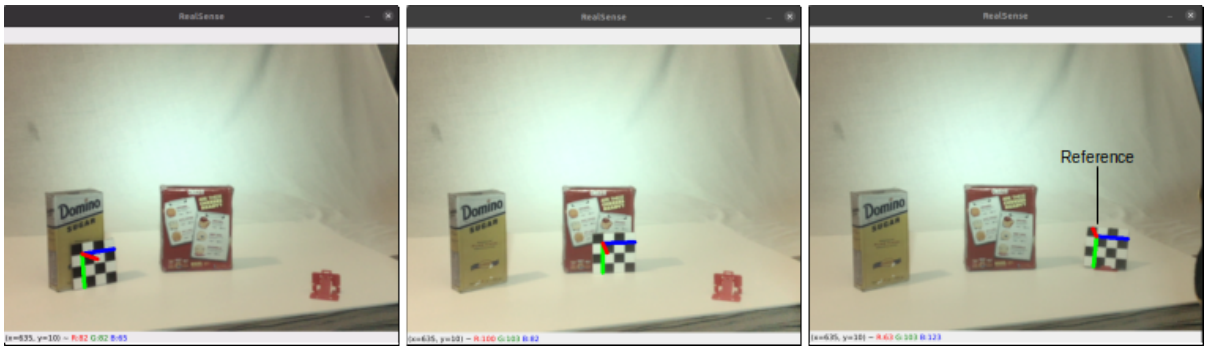

Figure 7.2: Ground truth poses measurements.

In total, 11 videos were recorded, containing images of the objects in different poses and from different perspectives. Following the same approach of [17], the resolution of the color sensor was set to $1280 * 720$

[1] Perspective-n-Point

and then cropped to $640 * 480$, compromising field of view in favor of higher resolution. The resolution of the depth sensor was set to its maximum, $1024 * 768$. It is worth noting that this is a limited dataset, as it contains a small amount of videos of only two objects, but this is exactly the goal of this experiment, which is to evaluate how simple it is to train a 6DoF pose estimator when it does not generalize to our application.

7.2.5 Training and Evaluation

Once the dataset was obtained, the training process could start. Both implemented versions of DenseFusion were trained with the new dataset. The models were trained for 10 epochs during 5 hours. This step was performed in the exact same manner as it was done in Chapter 4, but with one exception, since we are using pre-trained weights (Transfer learning), the learning rate should be set to a lower value than the values used while training with the YCB-Video dataset, but at the same time, it should be slightly larger than the value used on the last epoch to allow the network weights to find another optimal value. Hence, the chosen learning rate $25 * 10^{-}6$.

For training, 9 videos were used, and the other 2 were used for evaluation.

7.3 Results

In the beginning, the main goal of this experiment was to evaluate how well do the implemented models generalize to a real-world scenario. However, the results were not as good as expected. The models were tested in real-time, and only the translation of the objects in the scene were well estimated, while the rotation was incorrect as shown in figure 7.3. For this reason, the accuracy of the model in terms of the ADD(-S) metric for $w_{ADD} < 2cm$ was 0%. Even though it is not clearly known what could be causing this, one of the reasons might be that the handcrafted objects might have different dimensions from the original objects and, besides that, their reflectivity could also be different.

As a result a workaround solution was implemented, by creating a dataset in the laboratory, as described in the last section. The following accuracy and precision measurements were obtained on the videos left for evaluation of the created dataset. On the other hand, the inference time results were obtained online to understand whether the models can work in real-time or not. In the latter measurements, the scenario is the same as in the videos, with two objects in frame.

Figure 7.3: Poorly estimated pose before training on the case-study.

7.3.1 Accuracy Analysis

Table 7.1 shows the accuracy results obtained in the evaluation videos of the created dataset for the ADD(-S)$< 2cm$ and AUC metrics. As we can see, the accuracy of the implemented models is much better than without first adjusting the weights to the new scenario. Even though these results are not as good as those obtained in the YCB-Video Dataset, these prove that we can indeed adjust the model to a specific practical application and achieve satisfactory accuracy, at least, to this experimental setup. What is relevant on these data is that the pre-trained weights from Chapter 4 were used, and this allowed the model to adjust to a new scenario in relatively little time when compared with an untrained model.

	Original DenseFusion		Tweaked DenseFusion	
	AUC	<2cm	AUC	<2cm
cracker box	88,1	77,1	88,8	76,5
sugar box	88,6	72,8	91,1	82,3
AVG	88,4	75,0	89,9	79,4

Table 7.1: Implemented 6DoF pose estimators' performances on the case study dataset in the ADD(-s) metric for $\omega_{ADD} < 2cm$, and also AUC in terms of ADD(-S) metric for $0 < \omega_{ADD} < 10cm$.

It is also a noticeable fact that, in this evaluation session, the AUC metric surpassed the ADD(-S) results, as opposed to what we saw in the results from Chapter 4. However, this is not surprising if we consider the accuracy of the models for different w_{ADD} thresholds in tables 7.2 and 7.3. If we take the mean of those results, as the AUC metric requires, we end up with a higher value than the accuracy at $w_{ADD} < 2cm$.

	Original DenseFusion									
	<1cm	<2cm	<3cm	<4cm	<5cm	<6cm	<7cm	<8cm	<9cm	<10cm
cracker box	17,2	77,1	95,5	98,1	98,4	98,7	98,8	99,0	99,1	99,2
sugar box	26,4	72,8	94,1	98,1	98,7	99,0	99,0	99,1	99,1	99,1
AVG	21,8	75,0	94,8	98,1	98,6	98,9	98,9	99,1	99,1	99,2

Table 7.2: Implemented original DenseFusion on the case study dataset in the ADD(-S) metric for $0 < \omega_{ADD} < 10cm$.

	Tweaked DenseFusion									
	<1cm	<2cm	<3cm	<4cm	<5cm	<6cm	<7cm	<8cm	<9cm	<10cm
cracker box	27,5	76,5	93,7	97,2	98,2	98,7	98,9	99,0	99,1	99,2
sugar box	41,2	82,3	94,7	97,8	98,9	99,0	99,1	99,2	99,2	99,2
AVG	34,4	79,4	94,2	97,5	98,6	98,9	99,0	99,1	99,2	99,2

Table 7.3: Implemented tweaked DenseFusion on the case study dataset in the ADD(-S) metric for $0 < \omega_{ADD} < 10cm$.

7.3.2 Precision Analysis

In table 7.4, the results from the precision measurements using the proposed metric from Chapter 6 are presented. The first thing that stands out here is that the ground-truth poses of the created dataset got a precision score of 0. It might seem unexpected, but if we consider how these poses were obtained, we could not expect any other result but 0. The absolute poses were obtained using the known relative poses of the objects with the reference, so the relative pose between the objects in the scene will be constant during the whole video.

	Dataset	**DenseFusion Original**	**DenseFusion Tweak**
$\tau_{[R\|t]}$	0	0.1376	0.1300

Table 7.4: Precision measurements on the case-study.

This particularity highlights an important aspect related to the method used to generate the ground truth poses of the dataset that can be seen both as an advantage and a disadvantage. The relative poses of the objects are indeed constant during the whole video. Although this is advantageous in the sense that it achieves the best possible precision score for the ground truth poses, it sets a very high correlation of the objects' poses accuracy with that of the reference pose. So any error on the reference pose measurement will be enhanced when transforming it with the objects' relative poses. Therefore, when using this technique, one should be careful with the chosen 6DoF pose measurement tool.

If we now look at the precision scores obtained by the implemented models, we can see that these are

higher than those obtained in Chapter 6, which means that there are more jitter on the objects' estimated poses.

7.3.3 Inference Time Analysis

In Chapter 4, the implemented models were tested in terms of inference-time. Nevertheless, in that scenario, the frames were being uploaded from the machine's SSD. In this experiment, the computation time measurements are repeated, but this time, the measurements are obtained online, i.e., using frames that are being captured in real-time.

The results from this experiment are reported in table 7.5 in terms of frames per second.

	DenseFusion Original	**DenseFusion Tweak**
$\tau_{[R\|t]}$	22 FPS	21 FPS

Table 7.5: Inference-time measurements on the case-study.

As we can see, the algorithm works slightly faster when connected to an RGB-D sensor. These are very satisfactory results because we can now state that the models can indeed work in real-time. Still, these results correspond to a scenario where only two objects appear in the frame. If we had more, the inference time would increase, as this would require more computational effort.

7.4 Conclusion

In this study, the implemented models from Chapter 4 were put into practice. An experimental setup was built to emulate a real-world practical application and the models were evaluated in terms of generalization. Unfortunately, the results were rather unsatisfactory and, because of that, a new dataset was created, using images from the experimental setup. Once this was done, it was proven that it is straightforward to overcome this problem despite the poor initial generalization of the algorithms. A dataset can easily be created using the techniques above described and, with a pre-trained network, it takes very little time to train/adjust the network to the new practical setup, allowing it to achieve good accuracy and precision. Moreover, the models were evaluated in terms of inference-time, looking forward to answering the question: Can the models perform in real-time? And the answer is yes. Both implementations can perform at over 20 frames per second in a setup with two objects.

Chapter 8

Main Takeaways

In this thesis, four studies were performed in the 6DoF pose estimation field. The main goal was to cover all tools needed to build a 6DoF pose estimation model, use it in a practical scenario, evaluate its accuracy, precision, how it generalizes, and if it can work in real-time. All of these goals were accomplished.

In Chapter 3, a set of experiments were performed on three RGB-D sensors from Intel, where the Intel Realsense L515 outperformed the other two sensors. Furthermore, this sensor showed remarkable consistency under a wide range of distances, both in accuracy and precision. Therefore, it is fair to say that this sensor is an excellent fit for the object pose estimation field.

In Chapter 4, it was detailed how one can implement, train, and evaluate a machine learning based pose estimation model. Moreover, it was demonstrated how a simple change could be done to the DenseFusion's [3] original architecture, resulting in a $1,4\%$ accuracy improvement on the $ADD(-S) < 2cm$ metric.

Chapter 5 contains an experiment where a pose tracker is developed. The idea was to evaluate how TEASER++ [1] could be used to possibly improve the inference time of the models from Chapter 4. The results from this experiment were not satisfactory. Using the chosen techniques, it is not possible to improve DenseFusion's inference time.

Looking forward to measuring the precision/repeatability of both estimated poses and ground-truth poses, in Chapter 6 a metric was proposed. This metric outputs a score that quantifies how much jitter one can observe on the object poses in subsequent frames of a video.

Finally, in Chapter 7, it was demonstrated how a 6DoF pose estimation dataset could be done and used to train the implemented models from Chapter 4 using transfer learning. This experiment showed that

using the trained weights from the YCB-Dataset was not sufficient to achieve satisfactory accuracy on the practical application. However, after training it with the new dataset, a much more accurate model was obtained, proving that this problem could be easily solved.

Bibliography

[1] H. Yang, J. Shi, and L. Carlone, "TEASER: Fast and Certifiable Point Cloud Registration," no. c, pp. 1–42, jan 2020.

[2] X. Huang, G. Mei, J. Zhang, and R. Abbas, "A comprehensive survey on point cloud registration," mar 2021.

[3] C. Wang, D. Xu, Y. Zhu, R. Martin-Martin, C. Lu, L. Fei-Fei, and S. Savarese, "DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion," in CVPR, vol. 2019-June. IEEE, jun 2019, pp. 3338–3347.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," dec 2015.

[5] J. Howard and S. Gugger, Deep Learning for Coders with fastai and PyTorch. O'Reilly Media, 2020.

[6] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again," in 2017 IEEE International Conference on Computer Vision (ICCV). IEEE, oct 2017, pp. 1530–1538.

[7] M. Rad and V. Lepetit, "BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth," in 2017 IEEE International Conference on Computer Vision (ICCV). IEEE, oct 2017, pp. 3848–3856. [Online]. Available: http://ieeexplore.ieee.org/document/8237675/

[8] B. Tekin, S. N. Sinha, and P. Fua, "Real-Time Seamless Single Shot 6D Object Pose Prediction," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, jun 2018, pp. 292–301. [Online]. Available: https://ieeexplore.ieee.org/document/8578136/

[9] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image," in CVPR, vol. 2016-Decem. IEEE, jun 2016, pp. 3364–3372.

[10] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation," ECCV, vol. 16, no. 1, pp. 205–220, jul 2016.

[11] R. Rios-Cabrera and T. Tuytelaars, "Discriminatively Trained Templates for 3D Object Detection: A Real Time Scalable Approach," in ICCV. IEEE, dec 2013, pp. 2048–2055.

[12] K. K. S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski and N. Navab, Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes, ser. Lecture Notes in Computer

Science, A. Fusiello, V. Murino, and R. Cucchiara, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7584.

[13] T. Hodan, X. Zabulis, M. Lourakis, S. Obdrzalek, and J. Matas, "Detection and fine 3D pose estimation of texture-less objects in RGB-D images," in IROS, vol. 2015-Decem. IEEE, sep 2015, pp. 4421–4428.

[14] M. Gonzalez, A. Kacete, A. Murienne, and E. Marchand, "YOLOff: You Only Learn Offsets for robust 6DoF object pose estimation," no. c, feb 2020.

[15] R. Kouskouridas, A. Tejani, A. Doumanoglou, D. Tang, and T.-K. Kim, "Latent-Class Hough Forests for 6 DoF Object Pose Estimation," IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 1, pp. 119–132, feb 2016.

[16] "Kinect for xbox 360," https://marketplace.xbox.com/pt-PT/Product/Kinect-for-Xbox-360/66acd000-77fe-1000-9115-d8025858084b, 2021.

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

[18] C. Sahin, G. Garcia-Hernando, J. Sock, and T.-K. Kim, "A Review on Object Pose Recovery: from 3D Bounding Box Detectors to Full 6D Pose Estimators," jan 2020.

[19] C. Wang, R. Martín-Martín, D. Xu, J. Lv, C. Lu, L. Fei-Fei, S. Savarese, and Y. Zhu, "6-PACK: Category-level 6D Pose Tracker with Anchor-Based Keypoints," oct 2019.

[20] W. Kehl, F. Tombari, N. Navab, S. Ilic, and V. Lepetit, "Hashmod: A Hashing Method for Scalable 3D Object Detection," BMVC, pp. 36.1–36.12, jul 2016.

[21] S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige, "Going Further with Point Pair Features," ECCV, vol. 9907 LNCS, pp. 834–848, nov 2017.

[22] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3D object recognition," in CCVPR. IEEE, jun 2010, pp. 998–1005.

[23] D. Xu, D. Anguelov, and A. Jain, "PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation," nov 2017.

[24] M. Tian, L. Pan, M. H. Ang, and G. H. Lee, "Robust 6D Object Pose Estimation by Learning RGB-D Features," feb 2020.

[25] Z. Xu, K. Chen, and K. Jia, "W-PoseNet: Dense Correspondence Regularized Pixel Pair Pose Regression," dec 2019.

[26] N. Pereira and L. A. Alexandre, "MaskedFusion: Mask-based 6D Object Pose Estimation," pp. 1–17, nov 2019.

[27] W. Chen, J. Duan, H. Basevi, H. J. Chang, and A. Leonardis, "PointPoseNet: Point Pose Network for Robust 6D Object Pose Estimation," in WACV. IEEE, mar 2020, pp. 2813–2822.

[28] S. Sun, R. Liu, Q. Du, and S. Sun, "Selective Embedding with Gated Fusion for 6D Object Pose Estimation," Neural Processing Letters, vol. 51, no. 3, pp. 2417–2436, jun 2020.

[29] Y. Cheng, H. Zhu, C. Acar, W. Jing, Y. Wu, L. Li, C. Tan, and J.-H. Lim, "6D Pose Estimation with Correlation Fusion," sep 2019.

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite Andreas," pp. 3354–3361, 2012.

[31] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite," in CVPR, vol. 07-12-June. IEEE, jun 2015, pp. 567–576.

[32] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images," in ECCV, 2012, vol. 7576 LNCS, no. PART 5, pp. 746–760.

[33] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond PASCAL: A benchmark for 3D object detection in the wild," in WACV. IEEE, mar 2014, pp. 75–82.

[34] C. Rennie, R. Shome, K. E. Bekris, and A. F. De Souza, "A Dataset for Improved RGBD-Based Object Detection and Pose Estimation for Warehouse Pick-and-Place," IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 1179–1185, 2016.

[35] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D Object Pose Estimation Using 3D Object Coordinates," in Lecture Notes in Computer Science, 2014, vol. 8690 LNCS, no. PART 2, pp. 536–551.

[36] A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T. K. Kim, "Recovering 6D Object Pose and Predicting Next-Best-View in the Crowd," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, pp. 3583–3592, dec 2016.

[37] T. Hodan, P. Haluza, S. Obdrzalek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects," WACV, pp. 880–888, jan 2017.

[38] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother, "BOP: Benchmark for 6D Object Pose Estimation," Lecture Notes in Computer Science, vol. 11214 LNCS, pp. 19–35, aug 2018.

[39] H. Yang and L. Carlone, "A Polynomial-time Solution for Robust Registration with Extreme Outlier Rates," mar 2019.

[40] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in Robotics: Science and Systems V. Robotics: Science and Systems Foundation, jun 2009.

[41] Z. Gojcic, C. Zhou, J. D. Wegner, and A. Wieser, "The Perfect Match: 3D Point Cloud Matching with Smoothed Densities," nov 2018.

[42] R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," in 2009 IEEE International Conference on Robotics and Automation. IEEE, may 2009, pp. 3212–3217.

[43] F. Peng, Q. Wu, L. Fan, J. Zhang, Y. You, J. Lu, and J.-Y. Yang, "Street view cross-sourced point cloud matching and registration," in 2014 IEEE International Conference on Image Processing (ICIP). IEEE, oct 2014, pp. 2026–2030.

[44] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," dec 2019.

[45] Z. Yang, J. Z. Pan, L. Luo, X. Zhou, K. Grauman, and Q. Huang, "Extreme Relative Pose Estimation for RGB-D Scans via Scene Completion," dec 2018.

[46] L. Wang, J. Chen, X. Li, and Y. Fang, "Non-Rigid Point Set Registration Networks," apr 2019.

[47] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," in Proceedings. 1991 IEEE International Conference on Robotics and Automation. IEEE Comput. Soc. Press, pp. 2724–2729.

[48] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in Proceedings Third International Conference on 3-D Digital Imaging and Modeling. IEEE Comput. Soc, pp. 145–152.

[49] P. Besl and N. D. McKay, "A method for registration of 3-D shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239–256, feb 1992.

[50] S. Chang, C. Ahn, M. Lee, and S. Oh, "Graph-matching-based correspondence search for nonrigid point cloud registration," Computer Vision and Image Understanding, vol. 192, p. 102899, mar 2020.

[51] Bing Jian and B. C. Vemuri, "Robust Point Set Registration Using Gaussian Mixture Models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 8, pp. 1633–1645, aug 2011.

[52] H. Le, T.-T. Do, T. Hoang, and N.-M. Cheung, "SDRSAC: Semidefinite-Based Randomized Approach for Robust Point Cloud Registration without Correspondences," apr 2019.

[53] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions," mar 2016.

[54] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in 2008 10th International Conference on Control, Automation, Robotics and Vision. IEEE, dec 2008, pp. 643–650.

[55] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, sep 2008, pp. 3384–3391.

[56] Z. Dang, F. Wang, and M. Salzmann, "3D Registration for Self-Occluded Objects in Context," nov 2020.

[57] D. W. J. M. van de Wouw, G. Dubbelman, and P. H. N. de With, "Hierarchical 2.5-D Scene Alignment for Change Detection With Large Viewpoint Differences," IEEE Robotics and Automation Letters, vol. 1, no. 1, pp. 361–368, jan 2016.

[58] J. Kim, H. Kim, and J.-I. Park, "An Analysis of Factors Affecting Point Cloud Registration for Bin Picking," in 2020 International Conference on Electronics, Information, and Communication (ICEIC). IEEE, jan 2020, pp. 1–4.

[59] J. Vidal, C.-Y. Lin, X. Lladó, and R. Martí, "A Method for 6D Pose Estimation of Free-Form Rigid Objects Using Point Pair Features on Range Data," Sensors, vol. 18, no. 8, p. 2678, aug 2018.

[60] "Microsoft, "kinect for xbox 360"," https://marketplace.xbox.com/pt-PT/Product/Kinect-for-Xbox-360/66acd000-77fe-1000-9115-d8025858084b.

[61] F. Basso, M. Munaro, S. Michieletto, E. Pagello, and E. Menegatti, "Fast and Robust Multi-people Tracking from RGB-D Data for a Mobile Robot," 2013, pp. 265–276.

[62] Q. Wu, G. Xu, S. Zhang, Y. Li, and F. Wei, "Human 3D pose estimation in a lying position by RGB-D images for medical diagnosis and rehabilitation," in 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). IEEE, jul 2020, pp. 5802–5805.

[63] C. Xia, L. Wang, B.-K. Chung, and J.-M. Lee, "In Situ 3D Segmentation of Individual Plant Leaves Using a RGB-D Camera for Agricultural Automation," Sensors, vol. 15, no. 8, pp. 20 463–20 479, aug 2015.

[64] R. A. Newcombe, A. Fitzgibbon, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, and S. Hodges, "KinectFusion: Real-time dense surface mapping and tracking," in 2011 10th IEEE International Symposium on Mixed and Augmented Reality. IEEE, oct 2011, pp. 127–136.

[65] L. Sun, K. Yang, X. Hu, W. Hu, and K. Wang, "Real-Time Fusion Network for RGB-D Semantic Segmentation Incorporating Unexpected Obstacle Detection for Road-Driving Images," IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 5558–5565, oct 2020.

[66] "Intel® realsense™ depth camera sr305," https://www.intelrealsense.com/depth-camera-sr305/.

[67] "Orbbec astra series," https://orbbec3d.com/product-astra-pro/.

[68] "Intel® realsense™ depth camera d415," https://www.intelrealsense.com/depth-camera-d415/.

[69] "Orbbec astra stereo s u3," https://orbbec3d.com/astrastereos-2/.

[70] "Opencv ai kit : Oak—d," https://store.opencv.ai/products/oak-d.

[71] "Intel® realsense™ lidar camera l515," https://www.intelrealsense.com/lidar-camera-l515/.

[72] "Microsoft, "kinect for windows," https://developer.microsoft.com/pt-br/windows/kinect/.

[73] "Asus xtion 2," http://xtionprolive.com/asus-xtion2.

[74] G. Halmetschlager-Funek, M. Suchi, M. Kampel, and M. Vincze, "An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments," IEEE Robotics and Automation Magazine, vol. 26, no. 1, pp. 67–77, 2019.

[75] H. Sarbolandi, D. Lefloch, and A. Kolb, "Kinect range sensing: Structured-light versus time-of-flight kinect," Computer Vision and Image Understanding, vol. 139, pp. 1 – 20, 2015.

[76] C.-Y. Chiu, M. Thelwell, T. Senior, S. Choppin, J. Hart, and J. Wheat, "Comparison of depth cameras for three-dimensional reconstruction in medicine," Journal of Engineering in Medicine, vol. 233, no. 9, 2019.

[77] F. L. Siena, B. Byrom, P. Watts, and P. Breedon, "Utilising the Intel RealSense Camera for Measuring Health Outcomes in Clinical Research," Journal of Medical Systems, vol. 42, no. 3, pp. 1–10, 2018.

[78] Vit, Adar, Shani, and Guy, "Comparing RGB-D sensors for close range outdoor agricultural phenotyping," Sensors (Switzerland), vol. 18, no. 12, pp. 1–17, 2018.

[79] Jing, Changjuan, Potgieter, Johan, Noble, Frazer, Wang, and Ruili, "A comparison and analysis of RGB-D cameras' depth performance for robotics application," in 2017 24th International Conference on Mechatronics and Machine Vision in Practice, M2VIP 2017, vol. 2017-December, 2017, pp. 1–6.

[80] Anxionnat, Adrien, Voros, Sandrine, Troccaz, and Jocelyne, "Comparative study of RGB-D sensors based on controlled displacements of a ground-truth model 1," in 2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018, 2018, pp. 125–128.

[81] Zollhöfer, Michael, Stotko, Patrick, Görlitz, Andreas, Theobalt, Christian, M. Nießner, R. Klein, and A. Kolb, "State of the art on 3D reconstruction with RGB-D cameras," Computer Graphics Forum, vol. 37, no. 2, pp. 625–652, 2018.

[82] Cao, Yan-Pei, Kobbelt, Leif, Hu, and Shi-Min, "Real-time High-accuracy 3D Reconstruction with Consumer RGB-D Cameras," ACM Transactions on Graphics, vol. 1, no. 1, 2018.

[83] P.Rosin, Y.-K. Lai, L.Shao, and Y. Liu, Eds., RGB-D Image Analysis and Processing. Springer, 2019.

[84] A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, Eds., Consumer Depth Cameras for Computer Vision. Springer, 2013.

[85] P. E. J. Kanti V. Mardia, Directional Statistics, 2nd ed., ser. Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc., jan 1999.

[86] "Configurable and powerful ring lights for machine vision," https://www.effilux.com/en/products/ring/effiring, 2020.

[87] "Intel® realsense™ d400 series product family," https://www.intelrealsense.com/depth-camera-d415/, 2020.

[88] P. E. J. Kanti V. Mardia, Directional Statistics, 2nd ed., ser. Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc., jan 1999.

[89] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," nov 2014.

[90] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of Tricks for Image Classification with Convolutional Neural Networks," dec 2018.

[91] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," dec 2016.

[92] "Pytorch," https://pytorch.org/.

[93] "fast.ai, "making neural nets uncool again"," https://www.fast.ai/.

[94] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," dec 2014.

[95] B. Wen, C. Mitash, B. Ren, and K. E. Bekris, "se(3)-TrackNet: Data-driven 6D Pose Tracking by Calibrating Image Residuals in Synthetic Domains," jul 2020.

[96] D. Lowe, "Object recognition from local scale-invariant features," in Proceedings of the Seventh IEEE International Conference on Computer Vision. IEEE, 1999, pp. 1150–1157 vol.2.

[97] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," 2006, pp. 404–417.

[98] "Opencv," https://opencv.org/, 2021.

[99] "Teaser++: fast & certifiable 3d registration," https://github.com/MIT-SPARK/TEASER-plusplus, 2021.

[100] "Point-feature-histogram," https://github.com/stevenliu216/Point-Feature-Histogram, 2021.

[101] T. Hodaň, J. Matas, and Š. Obdržálek, "On evaluation of 6d object pose estimation," in Computer Vision – ECCV 2016 Workshops, G. Hua and H. Jégou, Eds. Cham: Springer International Publishing, 2016, pp. 606–619.

[102] "pyrelasense2," https://pypi.org/project/pyrealsense2/, 2021.

Chapter A

Detailed Results

Cam_{dist}	$\pm 10cm$	$\pm 35cm$	All Points
$D415_{0,5m}$	0,002	0,004	0,122
$D415_{0,75m}$	0,003	0,003	0,087
$D415_{1m}$	0,005	0,005	0,079
$D415_{1,25m}$	0,007	0,007	0,037
$D415_{1,5m}$	0,009	0,009	0,139
$D415_{1,75m}$	0,013	0,013	0,293
$D415_{2m}$	0,017	0,017	0,093
$D415_{2,5m}$	0,025	0,027	0,036
$D415_{3m}$	0,036	0,042	0,069
$D415_{3,5m}$	0,045	0,063	0,065
$L515_{0,5m}$	0,001	0,001	134,853
$L515_{0,75m}$	0,001	0,001	202,285
$L515_{1m}$	0,001	0,001	150,913
$L515_{1,25m}$	0,002	0,002	7,556
$L515_{1,5m}$	0,002	0,002	14,588
$L515_{1,75m}$	0,002	0,002	22,389
$L515_{2m}$	0,002	0,002	15,542
$L515_{2,5m}$	0,003	0,003	14,865
$L515_{3m}$	0,004	0,004	25,025
$L515_{3,5m}$	0,006	0,006	13,835
$SR305_{0,3m}$	0,003	0,036	113,994
$SR305_{0,4m}$	0,005	0,005	114,889
$SR305_{0,5m}$	0,008	0,008	0,008
$SR305_{0,75m}$	0,009	0,009	0,009
$SR305_{1m}$	0,005	0,005	0,006
$SR305_{1,25m}$	0,007	0,007	0,008
$SR305_{1,5m}$	0,008	0,008	0,011

Table A.1: Sensors' accuracy in terms of average root mean square point-to-plane distance error per image.

Cam_{dist}	$\overline{n_x}$	σ_{n_x}	$\overline{n_y}$	σ_{n_y}	$\overline{n_z}$	σ_{n_z}	$\overline{d}$	σ_d
$D415_{0,5m}$	3,9E-03	2,5E-04	4,1E-03	2,1E-04	1,0E+00	1,5E-06	-5,0E-01	1,1E-04
$D415_{0,75m}$	3,7E-03	3,0E-04	-8,8E-03	2,0E-04	1,0E+00	1,5E-06	-7,5E-01	2,2E-04
$D415_{1m}$	1,7E-02	7,2E-04	-1,3E-03	2,6E-04	1,0E+00	1,3E-05	-1,0E+00	1,6E-04
$D415_{1,25m}$	1,4E-02	1,6E-03	-2,7E-03	3,6E-04	1,0E+00	1,8E-05	-1,3E+00	3,1E-04
$D415_{1,5m}$	1,8E-02	8,7E-04	3,0E-03	4,3E-04	1,0E+00	1,6E-05	-1,5E+00	4,6E-04
$D415_{1,75m}$	2,2E-02	8,5E-04	3,4E-04	5,8E-04	1,0E+00	1,8E-05	-1,8E+00	4,9E-04
$D415_{2m}$	1,3E-02	2,9E-03	2,4E-03	8,6E-04	1,0E+00	2,9E-05	-2,0E+00	1,1E-03
$D415_{2,5m}$	1,0E-02	1,5E-03	-3,2E-03	9,3E-04	1,0E+00	1,4E-05	-2,5E+00	2,8E-03
$D415_{3m}$	1,4E-02	3,2E-03	-5,8E-04	1,1E-03	1,0E+00	3,3E-05	-3,0E+00	1,8E-03
$D415_{3,5m}$	6,6E-03	1,8E-03	1,8E-03	1,3E-03	1,0E+00	1,2E-05	-3,5E+00	3,2E-03
$L515_{0,5m}$	-4,6E-05	6,1E-04	-2,9E-03	2,3E-04	1,0E+00	7,9E-07	-5,0E-01	3,5E-04
$L515_{0,75m}$	1,8E-03	4,6E-04	-2,7E-03	1,6E-04	1,0E+00	7,1E-07	-7,5E-01	3,3E-04
$L515_{1m}$	-1,9E-03	6,1E-04	-5,7E-03	1,5E-04	1,0E+00	1,9E-06	-1,0E+00	3,5E-04
$L515_{1,25m}$	-6,8E-06	3,1E-04	2,1E-03	7,6E-05	1,0E+00	1,9E-07	-1,3E+00	3,1E-04
$L515_{1,5m}$	-3,1E-03	5,2E-04	-9,2E-04	1,9E-04	1,0E+00	1,7E-06	-1,5E+00	3,3E-04
$L515_{1,75m}$	1,8E-03	3,3E-04	3,1E-04	1,0E-04	1,0E+00	6,2E-07	-1,8E+00	3,1E-04
$L515_{2m}$	-8,0E-04	4,5E-04	-5,1E-04	1,7E-04	1,0E+00	4,4E-07	-2,0E+00	3,9E-04
$L515_{2,5m}$	3,2E-04	3,5E-04	-8,7E-04	1,4E-04	1,0E+00	1,7E-07	-2,5E+00	3,5E-04
$L515_{3m}$	-2,1E-03	3,3E-04	-3,9E-03	1,5E-04	1,0E+00	1,1E-06	-3,0E+00	3,3E-04
$L515_{3,5m}$	1,7E-03	4,0E-04	-6,8E-03	8,3E-05	1,0E+00	7,7E-07	-3,5E+00	3,5E-04
$SR305_{0,3m}$	4,4E-03	4,6E-03	-5,6E-03	1,0E-04	1,0E+00	2,7E-05	-3,0E-01	1,8E-04
$SR305_{0,4m}$	7,7E-03	5,2E-03	-8,2E-04	8,7E-05	1,0E+00	6,2E-05	-4,0E-01	2,3E-04
$SR305_{0,5m}$	1,7E-02	7,8E-03	5,2E-03	1,2E-04	1,0E+00	1,6E-04	-5,0E-01	2,7E-04
$SR305_{0,75m}$	-1,7E-02	1,3E-02	-1,3E-04	2,8E-04	1,0E+00	2,6E-04	-7,5E-01	6,3E-04
$SR305_{1m}$	-1,7E-02	7,3E-03	6,6E-03	1,8E-04	1,0E+00	1,0E-04	-1,0E+00	7,7E-04
$SR305_{1,25m}$	-2,7E-02	1,2E-02	-2,3E-03	5,0E-04	1,0E+00	2,6E-04	-1,3E+00	1,3E-03
$SR305_{1,5m}$	-2,3E-02	2,0E-02	-2,9E-03	1,1E-03	1,0E+00	5,8E-04	-1,5E+00	2,8E-03

Table A.2: Camera precision in terms of plane modelling consistency.

Cam_{dist}	$\overline{\phi}$	σ_ϕ	V	Cam_{dist}	$\overline{\phi}$	σ_ϕ	V
$D415_{0,5m}$	9,0E+01	1,5E-02	5,3E-08	$L515_{1,5m}$	9,0E+01	3,1E-02	1,5E-07
$D415_{0,75m}$	8,9E+01	9,2E-03	6,5E-08	$L515_{1,75m}$	9,0E+01	1,9E-02	5,9E-08
$D415_{1m}$	8,9E+01	4,1E-02	2,9E-07	$L515_{2m}$	9,0E+01	2,4E-02	1,2E-07
$D415_{1,25m}$	8,9E+01	8,5E-02	1,3E-06	$L515_{2,5m}$	9,0E+01	8,6E-03	7,3E-08
$D415_{1,5m}$	8,9E+01	4,9E-02	4,7E-07	$L515_{3m}$	9,0E+01	1,4E-02	6,5E-08
$D415_{1,75m}$	8,9E+01	4,9E-02	5,3E-07	$L515_{3,5m}$	9,0E+01	6,2E-03	8,3E-08
$D415_{2m}$	8,9E+01	1,6E-01	4,6E-06	$SR305_{0,3m}$	9,0E+01	1,5E-01	1,1E-05
$D415_{2,5m}$	8,9E+01	7,9E-02	1,6E-06	$SR305_{0,4m}$	9,0E+01	3,0E-01	1,4E-05
$D415_{3m}$	8,9E+01	1,7E-01	5,6E-06	$SR305_{0,5m}$	8,9E+01	4,2E-01	3,1E-05
$D415_{3,5m}$	9,0E+01	1,0E-01	2,5E-06	$SR305_{0,75m}$	8,9E+01	7,0E-01	9,0E-05
$L515_{0,5m}$	9,0E+01	1,4E-02	2,1E-07	$SR305_{1m}$	8,9E+01	3,2E-01	2,7E-05
$L515_{0,75m}$	9,0E+01	1,2E-02	1,2E-07	$SR305_{1,25m}$	8,8E+01	5,6E-01	7,4E-05
$L515_{1m}$	9,0E+01	1,8E-02	2,0E-07	$SR305_{1,5m}$	8,9E+01	1,0E+00	2,1E-04
$L515_{1,25m}$	9,0E+01	4,8E-03	5,2E-08	—	—	—	—

Table A.3: Camera precision in terms of plane normal vector angles standard deviation and spherical variance.

Method	ape	bvise	cam	can	cat	drill.	duck	egg.*	glue*	hp.	iron	lamp	phone	AVG	Refine.
Brach et al. [9]	98,8	99	99,7	99,7	99,1	100	96,2	99,7	99	98	99,9	99,5	99,6	99	RANSAC
Kehl et al.[10]	96,9	94,1	97,7	95,2	97,4	96,2	97,3	99,9	78,6	96,8	98,7	96,2	92,8	95,2	None
Cabrera et al. [11]	95	98,9	98,2	96,3	99,1	94,3	94,2	99,8	96,3	97,5	98,4	97,9	88,3	96,5	None
Linemod [12]	95,8	98,7	97,5	95,4	99,3	93,6	95,9	99,8	91,8	95,9	97,5	97,7	93,3	96,3	ICP
Hodan et al. [13]	93,9	99,8	95,5	95,9	98,2	94,1	94,3	100	98,0	88,0	97,0	88,8	89,4	94,9	PSO
Hashmod [20]	95,6	91,2	95,2	91,8	96,1	95,1	92,9	99,9	95,4	94,3	94,3	94,9	91,3	94,6	ICP
Hinters et al. [21]	98,5	99,8	99,3	98,7	99,9	93,4	98,2	98,8	75,4	98,1	98,3	96	98,6	96,4	ICP
DenseFusion [3]	79,5	84,2	76,5	86,6	88,8	77,7	76,3	99,9	99,4	79,0	92,1	92,3	88,0	86,2	None
DenseFusion [3]	92,3	93,2	94,4	93,1	96,5	87,0	92,3	99,8	100,0	92,1	97,0	95,3	92,8	94,3	NN
Meng Tian et al. [24]	85,0	95,6	91,3	95,2	93,6	82,6	88,1	99,9	99,6	92,6	95,9	94,4	93,6	92,9	None
Meng Tian et al.[25]	86,3	97,4	98,3	97,5	97,3	95,9	93,3	99,9	99,8	95,7	96,6	99,2	96,4	96,4	None
Meng Tian et al.[25]	92,8	99,6	99,0	99,3	99,0	97,8	96,2	99,9	99,9	97,3	98,6	99,8	98,3	98,2	NN
MaskedFusion[26]	92,2	98,4	98,0	97,4	97,8	95,6	94,0	99,6	100	97,3	97,1	99,0	98,8	97,3	NN
YOLOFF[14]	87,9	99,7	89,5	94,8	97,6	97,8	85,0	-	-	90,5	98,0	98,1	97,2	94,2	None
YOLOFF[14]	93,7	99,8	98,0	99,3	99,2	99,5	94,7	-	-	98,1	99,6	99,0	98,3	98,1	ICP
PointPoseNet[27]	97,9	99,6	98,5	99,4	99,3	97,5	96,1	97,9	100	97,8	99,4	99,1	98,9	98,4	None
Shantong Sun et al.[28]	93,2	97,8	96,7	98,1	98,4	94,7	94,8	99,8	99,8	94,2	97,3	98,7	97,5	97,0	None
Yi Cheng et al. [29]	95,6	96,9	97,9	96,0	97,8	95,6	95,7	99,9	99,7	96,7	97,8	97,0	97,0	97,2	None

Table A.4: Methods' performances on the LINEMOD dataset in the ADD(-S) metric for $k_m = 0, 1$. The last column is the refinement step. The objects marked with a "*" are symmetric.

	DenseFusion [3]		DenseFusion (rf) [3]		Meng Tian et al. [24]	W-PoseNet[25]		W-PoseNet (rf) [25]		MaskedFusion[26]		Shantong Sun et al.[28]		Yi Cheng et al. [29] Fuse_V2	
	AUC	<2cm	AUC	<2cm	AUC	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm
master chef can	95,2	100,0	96,4	100,0	93,9	69,2	65,9	72,0	68,6	95,5	100,0	95,6	100,0	92,5	98,7
cracker box	92,5	99,3	95,5	99,5	92,9	87,8	90,0	91,3	93,7	96,7	99,8	95,6	99,7	95,5	98,6
sugar box	95,1	100,0	97,5	100,0	95,4	91,5	98,5	95,1	99,8	98,1	100,0	97,8	100,0	96,7	99,9
tomato soup can	93,7	96,9	94,6	100,0	93,3	87,4	84,2	88,9	84,6	94,3	96,9	94,4	96,9	92,0	95,8
mustard bottle	95,9	100,0	97,2	100,0	95,4	93,4	100,0	96,5	100,0	98,0	100,0	97,7	100,0	94,8	97,5
tuna fish can	94,9	100,0	96,6	100,0	94,9	77,0	55,9	78,8	61,4	96,9	99,7	97,2	100,0	88,9	84,2
pudding box	94,7	100,0	96,5	93,1	94,0	91,8	98,1	94,5	100,0	97,3	100,0	96,3	100,0	93,2	98,6
gelatin box	95,8	100,0	98,1	100,0	97,6	94,6	100,0	96,0	100,0	98,3	100,0	98,5	100,0	95,7	100,0
potted meat can	90,1	93,1	91,3	100,0	90,6	79,0	77,8	82,6	80,4	89,6	94,2	91,4	92,7	86,2	83,9
banana	91,5	93,9	96,6	100,0	91,7	87,9	87,3	92,8	98,9	97,6	100,0	97,0	99,7	92,6	98,9
pitcher base	94,6	100,0	97,1	98,8	93,1	92,0	100,0	95,0	100,0	97,7	100,0	97,9	100,0	95,4	98,4
bleach cleanser	94,3	99,8	95,8	100,0	93,4	85,2	77,4	89,5	89,0	95,4	99,4	95,1	99,7	89,0	86,2
bowl*	86,6	69,5	88,2	98,8	92,9	86,2	49,8	87,7	93,6	89,6	95,4	87,9	99,8	86,1	94,3
mug	95,5	100,0	97,1	100,0	96,1	84,9	79,7	88,2	90,3	97,1	100,0	96,9	100,0	93,5	94,8
power drill	92,4	97,1	96,0	98,7	93,3	91,1	98,8	93,6	99,5	96,7	99,5	96,9	99,3	82,9	84,8
wood block*	85,5	93,4	89,7	94,6	87,6	86,3	96,3	87,0	97,5	91,8	100,0	92,4	99,6	92,3	99,6
scissors	96,4	100,0	95,2	100,0	95,7	91,5	99,5	90,7	97,8	92,7	99,9	94,1	100,0	90,2	89,5
large marker	94,7	99,2	97,5	100,0	95,6	90,9	96,3	92,5	99,4	97,5	99,9	97,4	99,9	93,9	99,9
large clamp*	71,6	78,5	72,9	79,2	75,4	71,4	74,0	70,8	79,2	71,9	78,7	72,9	78,8	70,3	76,7
extra large clamp*	69,0	69,5	69,8	76,3	73,0	68,0	60,4	69,6	72,7	71,4	75,9	74,9	75,5	69,5	74,5
foam brick*	92,4	100,0	92,5	100,0	94,2	92,5	100,0	92,9	100,0	94,3	100,0	95,6	100,0	94,6	100,0
AVG	91,2	95,3	93,1	96,8	91,8	85,7	85,2	87,9	90,8	93,3	97,1	93,5	96,9	89,8	93,1

Table A.5: Methods' performances on the YCB-Video dataset in the ADD(-S) metric for $\omega_{ADD} < 2cm$, and also AUC in terms of ADD(-S) metric for $0 < \omega_{ADD} < 10cm$. The "(rf)" next to some methods' names stands for refinement. Objects with a "*" next to them are symmetric.

Chapter B

Hyperparameter selection of Chapter 4

In Chapter 4, the DenseFusion's implementation details are presented. However, as it can be seen, many hyperparameter choices lack explanation. These hyperparameters are the number of layers, number of filters used in the convolutional layers, the number of activations on the linear layers, the learning rate, the images' sizes, and the number of extracted features from point clouds and RGB images.

Having in mind that these parameter's choice was needed before the model was trained, the large size of the dataset (400+Gb), and how inefficient it was to perform a complete study on these parameters on the whole dataset. All the values were chosen by empirical evaluation on a small portion of the dataset while developing the model. Hence, the results of these small experiments are not reported in this work. Not only because the acquired data is very inaccurate (due to the small dataset), but also because of the large amount of performed experiments and how difficult it is to actually report this.

Obviously, the problem with this approach is that these choices are not guaranteed to be optimal. At least, it is not possible to prove even if they were. Once the model is trained, it is impossible to change the network architecture, such as the number of layers and number of activations.

Still, it is possible to explain the theory behind these choices. It is a rule of thumb in most machine learning models that the larger the network is, the better. The universal aproximation theorem easily explains this, as it states that any function can be approximated with the right amount of layers in a deep neural network. So, the first criterion that can be defined to choose these parameters is making these values as large as possible, namely, the number of layers and number of activations on the linear layers and filters on convolutional layers. The primary tradeoff of setting these values too high is the increased memory usage and computational intensity that directly affects inference time. This rule also applies when choosing the size of the images and extracted features. The more information is fed to the network, the better but,

again, compromising inference time and memory usage.

One hyperparameter that does not share this criterion is the learning rate. In actuality, the initial learning rate used in Chapter 4 $(0, 0001)$ was set following recommendations from other works. However, if it was not, the criterion that one can use to choose it is to set it as high as possible before training gets unstable, i.e., the training does not follow a smooth decreasing pattern.

Though most of the parameters can not be changed after training, there is still one that can: the number of extracted features. When choosing this value, one should consider three main factors of significant impact: accuracy, inference time, and memory usage.

When training, inference time is not important, as the primary goal in this step is to make the training as efficient as possible and hopefully achieve the best results in terms of accuracy. Because of that, the inference time should not be accounted for in training as this parameter can be changed later. Hence, in training, the major tradeoff factor to a large number of features is memory usage. In each training step, both images and network weights are passed to the GPU's RAM, so the chosen resolution of the images and the size of the network will define how much memory is used. The number of extracted features from the images was chosen to be 1536, which was the largest value possible without exceeding the 11Gb memory limit of the RTX 2080ti. The resolution of the cropped images is $192 * 128$, which were the least common multiples of 32 [1] that kept the original aspect ratio of the images.

After training, the number of features can be reduced or even augmented because in evaluation, the gradients do not need to be computed, and thus there is more available memory. So, the following study was performed, where the impact of the number of extracted features is measured in terms of accuracy and inference time. The number of extracted features was increased in steps of size 128, from 128 to 2432 which was the maximum possible value to set without overflowing the memory.

As shown in table B.1, the impact of the number of extracted features is quite small, from 128 to 2432 features, the change in accuracy is only 6%, and the inference time did not even change. These results are interesting, as one may expect to observe a larger influence of this parameter on the model's overall accuracy. However, these results show how robust the model is and, since a smaller number of extracted features can be used while maintaining good accuracy, it leaves more room for experiments without overflowing the GPU's memory. Still, it is essential to highlight that these are only results from evaluation. It is not known what the impact of this parameter in training is. Unfortunately, since each training epoch takes about 10 hours, it is unfeasible to perform this study on training.

[1] The RGB feature extraction block used in the DenseFusion's architecture downsamples the images by $1/32$

$N_{Features}$	$ADD(-S) < 0,02$	AVG time	STD Time
128	89,3	0,0078	0,0006
256	94,3	0,0078	0,0006
384	95,1	0,0077	0,0006
512	95,4	0,0076	0,0006
640	95,6	0,0077	0,0007
768	95,7	0,0076	0,0006
896	95,8	0,0075	0,0006
1024	95,9	0,0074	0,0006
1152	95,9	0,0075	0,0006
1280	96,1	0,0075	0,0006
1408	95,9	0,0074	0,0006
1536	96,0	0,0075	0,0006
1664	96,0	0,0075	0,0006
1792	96,1	0,0074	0,0006
1920	96,0	0,0075	0,0006
2048	96,0	0,0075	0,0006
2176	96,1	0,0075	0,0006
2304	96,0	0,0075	0,0006
2432	96,1	0,0075	0,0006

Table B.1: Tweaked DenseFusion's behavior under different number of extracted features. This results were obtained on the evaluation key-frames from the YCB-Video dataset. The accuracy results are in percentage and the times are in seconds.

Chapter C

Publications

C.1 Intel RealSense SR305, D415 and L515: Experimental Evaluation and Comparison of Depth Estimation

This paper was published at the VISIGRAPP 2021 International Conference, and it is appended on the following pages.

Intel RealSense SR305, D415 and L515: Experimental Evaluation and Comparison of Depth Estimation

Francisco Lourenço[1][a], Helder Araujo [1][b]
[1]*Institute of Systems and Robotics, Department of Elec. and Comp. Eng.,University of Coimbra, Coimbra, Portugal*
francisco.rlourenco@isr.uc.pt, helder@isr.uc.pt

Keywords: RGB-D cameras, Experimental Evaluation, Experimental Comparison

Abstract: In the last few years, Intel has launched several low-cost RGB-D cameras. Three of these cameras are the SR305, the L415, and the L515. These three cameras are based on different operating principles. The SR305 is based on structured light projection, the D415 is based on stereo based also using the projection of random dots, and the L515 is based on LIDAR. In addition, they all provide RGB images. In this paper, we perform an experimental analysis and comparison of the depth estimation by the three cameras.

1 INTRODUCTION

Consumer-level RGB-D cameras are affordable, small, and portable. These are some of the main features that make these types of sensors very suitable tools for research and industrial applications, ranging from practical applications such as 3D reconstruction, 6D pose estimation, augmented reality, and many more (Zollhöfer et al., 2018). For many applications, it is essential to know how accurate and precise an RGB-D camera is, to understand which sensor best suits the specific application (Cao et al., 2018). This paper aims to compare three models of RGB-D cameras from Intel, which can be useful for many users and applications.

The sensors are the RealSense SR305, D415, and L515. Each sensor uses different methods to calculate depth. The SR305 uses coded light, where a known pattern is projected into the scene and, by evaluating how this pattern deforms, depth information is computed. The D415 uses stereo vision technology, capturing the scene with two imagers and, by computing the disparity on the two images, depth can be retrieved. Finally, the L515 that measures time-of-flight, i.e., this sensor calculates depth by measuring the delay between light emission and light reception.

Several different approaches can be used to evaluate depth sensors (P.Rosin et al., 2019),(Fossati et al., 2013). In this case, we focused on accuracy and repeatability. For this purpose, the cameras were evaluated using depth images of 3D planes at several distances, but whose ground truth position and orientation were not used as we don't know them. Accuracy was measured in terms of point-to-plane distance, and precision was measured as the repeatability of 3D model reconstruction, i.e., the standard deviation of the parameters of the estimated 3D model (in this case, a 3D plane). We also calculated the average number of depth points per image where the cameras failed to calculate depth (and their standard deviation), and the number of outliers per image (points for which the depth was outside an interval). Moreover, we employ directional statistics (Kanti V. Mardia, 1999) on the planes' normal vectors to better illustrate how these models variate.

2 RELATED WORK

Depth cameras and RGB-D cameras have been analyzed and compared in many different ways. In (Halmetschlager-Funek et al., 2019) several parameters of ten depth cameras were experimentally analyzed and compared. In addition to that, an analysis of the cameras' response to different materials, noise characteristics, and precision were also evaluated. A comparative study on structured light and time-of-flight based Kinect cameras is done in (Sarbolandi et al., 2015). In (Chiu et al., 2019) depth cameras were compared considering medical applications and their specific requirements. Another comparison for medical applications is performed in (Siena et al.,

[a] https://orcid.org/0000-0001-7081-5641
[b] https://orcid.org/0000-0002-9544-424X

2018). A comparison for agricultural applications is performed in (Vit et al., 2018). Analysis for robotic applications is performed in (Jing et al., 2017). In (Anxionnat et al., 2018) several RGB-D sensors are analyzed and compared based on controlled displacements, with precision and accuracy evaluations.

3 METHODOLOGY

3.1 Materials

As aforementioned, the sensors used in this evaluation employ different depth estimation principles, which yields information about the sensor's performance and how these technologies compare to each other (for the specific criteria used).

The SR305 uses coded light, the D415 uses stereo vision, and the L515 uses LIDAR. The camera specifications are represented on table 1. The three cameras were mounted on the standard tripod.

To ensure constant illumination conditions, a LED ring (eff, 2020) was used as the only room illumination source.

Table 1: Sensors resolution (px*px) and range (m).

Sensor	SR305	D415	L515
Depth	640x480	1280x720	1024x768
Color	1920x1080	1920x1080	1920x1080
Range	[0.2 1.5]	[0.3 10]	[0.25 9]

Note that the values on table 1 are upper bounds, meaning that the specifications may vary for different sensors' configurations. It is also important to mention that the D415 range may vary with the light conditions.

3.2 Experimental setup

Each camera was mounted on a tripod and placed at a distance d of a wall. The wall is white and covers all the field of view of the cameras. The optical axes of the sensors are approximately perpendicular to the wall. Placed above the camera is the light source, above described. The light source points to the wall in the same direction as the camera. For practical reasons, the light source is slightly behind the camera so that the camera does not interfere with the light. A laptop is placed behind the camera where the camera software is executed and where the images are stored. All the experiments took place at night, avoiding any unwanted daylight. Hence, the room's light was kept constant between experiments and always sourced by the same element.

The camera, light source, and laptop were placed on top of a structure. We wanted everything to be high relative to the ground to ensure that the sensors captured neither floor nor ceiling. For each distance at which the cameras were placed, 100 images were acquired. The distances for which both D415 and L515 were tested are 0.5m, 0.75m, 1m, 1.25m, 1.5m, 1.75m, 2m, 2.5m, 3m, and 3.5m. The furthest distance was the maximum distance for which neither floor nor ceiling appeared on the images. The SR305 was tested at 0.3m, 0.4m, 0.5m, 0.75m, 1m, 1.25m, and 1.5m. In this case, the furthest distance is the maximum specified range for the SR305 sensor.

The experiments started at the closest distance. The sensors were switched right after the other sequentially. After all the images were obtained at that distance, all the structure was moved away from the wall by the aforementioned intervals. The structure moved approximately perpendicularly to the wall.

For the D415 and the L515 sensors, we used custom configurations. For the SR305 we used the default configuration. These configurations were the same as those of table 1.

3.3 Software

To deal with the sensors, the Intel RealSense SDK 2.0 was used. The Intel RealSense Viewer application was used to check the sensors' behavior right before each execution, to check for the direction of the optical axis and distance. All the other tasks were executed using custom code and the librealsense2 library. These tasks include both image acquisition and storing, camera configuration, and point cloud generation. This part of the work was executed using Ubuntu. All the statistical evaluation was performed in MatLab, on Windows 10.

3.4 Experimental Evaluation

3.4.1 Performance

The performance of the sensors was measured in two ways. First, we calculated the average number of points for which the sensor failed to measure depth and the standard deviation of the same number of points. Then we do the same for outliers.

Whenever the Intel RealSense SDK 2.0 and camera fail to measure the depth at some point, the corresponding depth is defined as zero. Hence, all we do here is to count pixels in the depth image with a depth equal to zero.

Depth values also contain outliers. Outliers can be defined in several ways. In this case, we considered

as an outlier every point with a depth value differing 10cm from the expected distance, given the specific geometric configuration and setup.

As described in the Intel RealSense D415 product DataSheet (D41, 2020), the D415 sensor has an invalid depth band, which is a region in the depth image for which depth cannot be computed.

The coordinate system of the left camera is used as the reference coordinate system for the stereo camera. The left and right cameras have the same field of view. However, due to their relative displacement, there is an area in the left image for which it is impossible to compute disparities since the corresponding 3D volume is not visible in the right camera. It results in a non-overlap region of the left and right cameras for which it is impossible to measure depth. This region appears in the image's leftmost area and is illustrated in figure 1. The total number of pixels in the invalid depth band can be calculated in pixels as follows:

$$InvalidDepthBand = \frac{VRES * HRES * B}{2 * Z * tan(\frac{HFOV}{2})} \quad (1)$$

Where $VRES$ and $HRES$ stand for vertical and horizontal resolution respectively (720px and 1280px), B is the baseline $\rightarrow$ 55mm, Z is the distance of scene from the depth module $\rightarrow d$ and $HFOV$ is the horizontal field of view $\rightarrow 64°$.

Bearing that in mind, the pixels in the invalid depth band were ignored in our calculations.

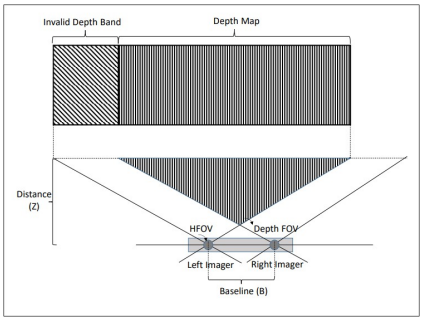

Figure 1: Invalid Depth Band.

3.4.2 Plane Fitting

Point clouds were first obtained using the depth data, the image pixel coordinates, and the camera intrinsics. This is possible because we have depth information, letting the coordinate z be equal to the measured depth at that point, i.e., $z = d_m$ the following equations can be applied:

$$x = z * \frac{u - pp_x}{f_x} \quad (2) \qquad y = z * \frac{v - pp_y}{f_y} \quad (3)$$

Where (u, v) are the pixel coordinates, (pp_x, pp_y) are the coordinates of the principal point, f_x and f_y are the focal lengths in pixel units.

The point clouds correspond to a wall. Thus it is possible to fit a plane to the data.

Since we handle ourselves the outliers, we performed the plane equation's estimation using standard least-squares regression, employing the singular value decomposition, instead of robust approaches such as RANSAC.

The model we want to be regressed to the point clouds is the general form of the plane equation:

$$x * n_x + y * n_y + z * n_z - d = 0 \quad (4)$$

Where $(n_x, n_y.n_z)$ stands for the unit normal, and d stands for distance from the plane to the origin.

If we now build a $n * 4$ matrix from the point clouds with n points, which we will denote as matrix P. We can rewrite equation 4:

$$0 = \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 & -1 \\ x_2 & y_2 & z_2 & -1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & -1 \end{bmatrix}}_{P} \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} \quad (5)$$

By computing the singular value decomposition on matrix P as:

$$P = U * \Sigma * V' \quad (6)$$

We can now use the values of the column of matrix V that corresponds to the smallest eigenvalue in matrix Σ, as the parameters n_x^*, n_y^*, n_z^* and d^* of the plane that fit that point cloud. Then, we normalize the plane's normal vector, which will become handy in further calculations and recover the true distance in meters of the plane from the sensor.

3.4.3 Accuracy

For the accuracy analysis, we compute the point-to-plane distance. For each point of the point cloud, we use the fitted plane equation to compute the errors, i.e., the point-to-plane distance. We then calculate the average root mean square error for each set of 100 images as an accuracy measurement. For the sake of comparison, we perform this computation for two different threshold values for outlier rejection ($\pm 10cm$ and $\pm 35cm$), another one where we use all points with the measured depth.

3.4.4 Precision

In this work, precision is measured as per image plane consistency, i.e., how the plane model changes in between images of the same sensor at the same distance.

As neither the scene nor the sensor change while taking the pictures, we could expect the models to be the exact same if we had an ideal sensor. Thus, by measuring the standard deviation of the plane model parameters in between images, we might be able to better understand how consistent the sensors are with their measurements and how this consistency varies with the distance.

Additionally, we also transform the plane's normal vector into spherical coordinates, where we can perform analysis of directional statistics as all the normals are distributed on a spherical surface. Specifically, the circular mean and standard deviation of angles θ and ϕ, and the spherical variance of the normal vectors. Since $\vec{n_i}$ is unitary, its norm ρ is 1.

Let θ and ϕ be the azimuth and altitude angles of $\vec{n_i}$:

$$\theta_i = \arctan \frac{n_{y_i}}{n_{x_i}} \quad (7) \qquad \phi_i = \arctan \frac{n_{z_i}}{\sqrt{n_{x_i}^2 + n_{y_i}^2}} \quad (8)$$

As in (Kanti V. Mardia, 1999), the circular mean of the angles above can be computed as follows:

$$\overline{\theta} = \arctan \frac{\sum_{i=1}^{n} \sin \theta_i}{\sum_{i=1}^{n} \cos \theta_i} \quad (9)$$

$$\overline{\phi} = \arctan \frac{\sum_{i=1}^{n} \sin \phi_i}{\sum_{i=1}^{n} \cos \phi_i} \quad (10)$$

Now, to show how the spherical variance is computed, we need to introduce vector $\overline{\vec{n}}$, which is the vector whose components are the mean of each component of $\vec{n}$. If we now compute the norm of $\overline{\vec{n}}$ and call it $\overline{R}$, the spherical variance is calculated as follows:

$$V = 1 - \overline{R} \quad (11)$$

4 Results

4.1 Performance analysis

In tables 2, 3 (In Appendix) and figure 2, we show the results for the average number of failed points and the standard deviation of the number of failed points per image (points where the sensor could not compute depth). As it can be verified, the L515 sensor outperforms both D415 and SR305. This sensor not only showed to be capable of estimating more depth data relative to its resolution, but the results also show that that number (of failed points) remains almost constant up until 2 meters of distance. This can be explained by the fact that this sensor uses LIDAR technology, which is more robust than the stereo vision and coded light approaches since LIDAR directly computes depth.

On the other hand, the D415 sensor shows much better results than the SR305.

A relevant detail of this performance measurement is that the standard deviation of the number of failed depth points for the D415 is not strongly dependent with the distance, whereas that does not seem to be the case for the L515.

Tables 4, 5 (In Appendix) and figure 3 include the results for the number of outliers. Just as it happened in terms of the average number of failed depth points, the data for the D415 camera show an increase in the average number of outliers as the distance increases. On the other hand, the number of outliers for the SR305 seems to decrease with increasing distances. These results are quite different from those presented in table 2 (In Appendix). The main reason for these results is probably because the SR305 camera is estimating a relatively small number of depth points at higher distances, therefore decreasing the probability of outlier occurrences.

To better illustrate this, we show in figure 4 a sample image taken with the SR305 sensor at 1.5 meters of distance, where it is notorious the small amount of points for which this sensor is measuring depth (the dark region represents points where the depth was not computed).

On the other hand, for the case of the L515, the number of outliers is essentially independent of the distance. Even though there is some fluctuation in the numbers, the variation is relatively small. Considering the standard deviation of the number of outliers per image obtained with the L515 at 0.75, 1.5, and 1.75 meters, we can see that it is zero, meaning that the total number of outliers per image stayed constant over the 100 images. This led us to determine the pixels where L515 generated outliers. We found out that the miscalculated points always correspond to the same pixels from the image's leftmost column. We believe that this may be happening to an issue with the camera used in the experiments, which requires further investigation.

4.2 Accuracy analysis

The results of the accuracy study are represented on table 6 (In Appendix) and in figure 5, where we plot the point-to-plane RMSE distances for the three cameras, taking into account only points whose distances

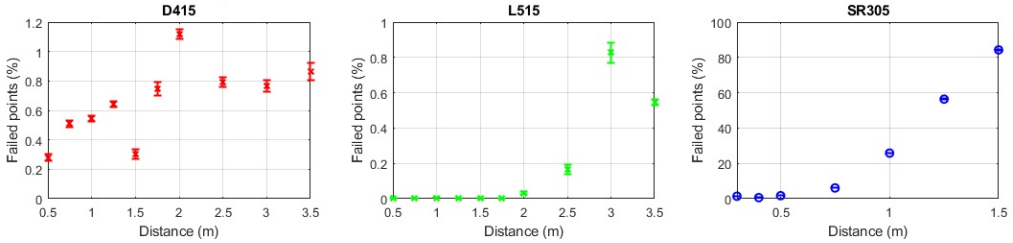

Figure 2: Failed Points - D415 vs L515 vs SR305.

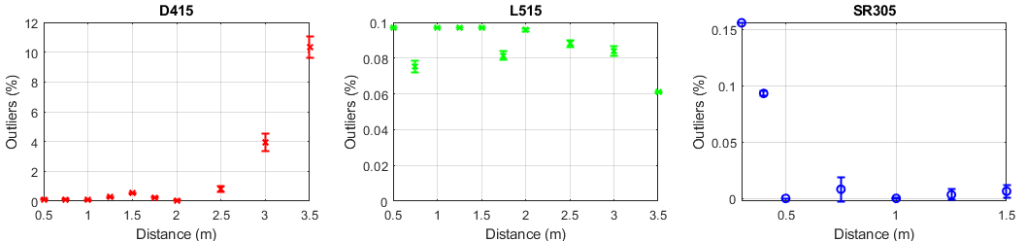

Figure 3: Outliers - D415 vs L515 vs SR305.

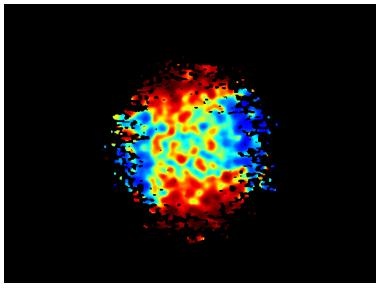

Figure 4: SR305 depth image at 1,5m.

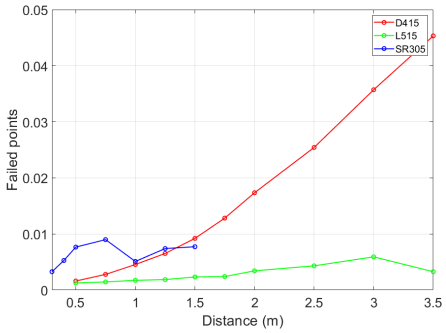

Figure 5: Point-to-plane RMSE D415 vs L515 vs SR305 - $\pm 10cm$.

from the expected distance are within 10cm.

Again, the sensor that achieves the best results is the L515. It not only has the lowest average root mean square error per image, but it also shows to be the least sensitive to distance. It should be mentioned that the image acquisition conditions for the L515 were close to being optimal since the images were acquired indoors, without daylight illumination.

In the case of camera D415, its RMSE follows a smooth increasing pattern as it goes further from the wall. On the other hand, the RMSE for the SR305 camera does not vary as smoothly with distance. The RMSE values for this sensor increase until 0.75m and then start to decrease until 1m. In fact, 1 meter is the distance for which the SR305 is optimized (SR3, 2020), therefore one should expect this sensor to work better within this range.

4.3 Precision analysis

The results show that the camera L515 is significantly more consistent than the other sensors. The results from tables 7 and 8 (In Appendix), show L515 to be more precise in terms of 3D estimation. It is noticeable how this sensor seems to be very consistent between pictures and for different distances.

In table 8 (In Appendix) we show the directional statistics results. The values for the angle θ frequently change in a non-systematic way. This happens because, as ϕ gets closer to 90°, the components n_x and n_y of the normal vector get closer to zero, which will lead to large variations of the angle θ. For this reason, we omit the azimuth calculations.

The spherical variation behaves just as expected, showing again that the L515 sensor is the most stable and that the measurements from the other two are more distance sensitive.

For ease of comprehension of our precision results, we plot on figures 6 and 7 the standard deviation of parameter d of the plane for all cameras at all distances, and also the spherical variation.

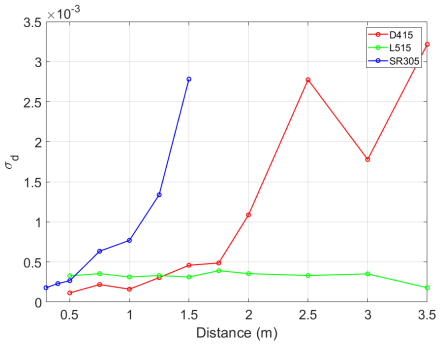

Figure 6: Parameter d standard deviation.

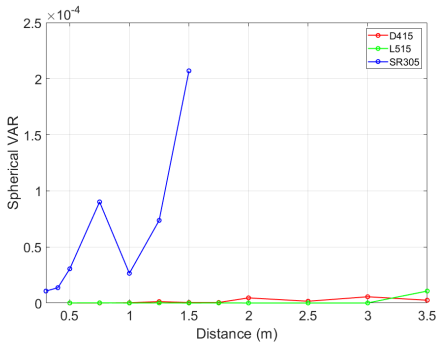

Figure 7: Spherical Variation.

5 CONCLUSION

In this paper, we described a set of experiments performed to compare the depth estimation performance of three RGB-D cameras from Intel, namely the SR305, the D415, and the L515. In general, the results show that the L515 is more accurate and precise than the other two while also providing more stable and consistent measurements in the specific environmental conditions of the experiments (indoors with controlled and stable illumination).

6 ACKNOWLEDGEMENTS

This work was partially supported by Project COMMANDIA SOE2/P1/F0638, from the Interreg Sudoe Programme, European Regional Development Fund (ERDF), and by the Portuguese Government FCT, project no. 006906, reference UID/EEA/00048/2013.

REFERENCES

(2020). Configurable and powerful ring lights for machine vision. https://www.effilux.com/en/products/ring/effiring. (Acessed: 05-10-2020).

(2020). Intel® realsense™ d400 series product family. https://www.intelrealsense.com/depth-camera-d415/. (Acessed: 01-10-2020).

(2020). Intel® realsense™ depth camera sr305. https://www.intelrealsense.com/depth-camera-sr305/. (Acessed: 02-10-2020).

Anxionnat, Adrien, Voros, Sandrine, Troccaz, and Jocelyne (2018). Comparative study of RGB-D sensors based on controlled displacements of a ground-truth model 1. In *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, pages 125–128.

Cao, Yan-Pei, Kobbelt, Leif, Hu, and Shi-Min (2018). Real-time High-accuracy 3D Reconstruction with Consumer RGB-D Cameras. *ACM Transactions on Graphics*, 1(1).

Chiu, C.-Y., Thelwell, M., Senior, T., Choppin, S., Hart, J., and Wheat, J. (2019). Comparison of depth cameras for three-dimensional reconstruction in medicine. *Journal of Engineering in Medicine*, 233(9).

Fossati, A., Gall, J., Grabner, H., Ren, X., and Konolige, K., editors (2013). *Consumer Depth Cameras for Computer Vision*. Springer.

Halmetschlager-Funek, G., Suchi, M., Kampel, M., and Vincze, M. (2019). An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments. *IEEE Robotics and Automation Magazine*, 26(1):67–77.

Jing, C., Potgiete, J., Noble, F., and Wang, R. (2017). A comparison and analysis of rgb-d cameras'depth performance for robotics application. In *Proc. of the Int- Conf. on on Mechatronics and Machine Vision in Practice*.

Kanti V. Mardia, P. E. J. (1999). *Directional Statistics*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2nd edition.

P.Rosin, Lai, Y.-K., L.Shao, and Liu, Y., editors (2019). *RGB-D Image Analysis and Processing*. Springer.

Sarbolandi, H., Lefloch, D., and Kolb, A. (2015). Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer Vision and Image Understanding*, 139:1 – 20.

Siena, F. L., Byrom, B., Watts, P., and Breedon, P. (2018). Utilising the Intel RealSense Camera for Measuring Health Outcomes in Clinical Research. *Journal of Medical Systems*, 42(3):1–10.

Vit, Adar, Shani, and Guy (2018). Comparing RGB-D sensors for close range outdoor agricultural phenotyping. *Sensors (Switzerland)*, 18(12):1–17.

Zollhöfer, Michael, Stotko, Patrick, Görlitz, Andreas, Theobalt, Christian, Nießner, M., Klein, R., and Kolb, A. (2018). State of the art on 3D reconstruction with RGB-D cameras. *Computer Graphics Forum*, 37(2):625–652.

APPENDIX

Table 2: Average of failed points ratio per image by distance in meters.

d	D415	L515	d	SR305
0.5	0.2801%	0.0020%	0.3	1.3967%
0.75	0.5099%	0.0001%	0.4	0.6062%
1	0.5414%	0.0023%	0.5	1.6930%
1.25	0.6415%	0.0001%	0.75	6.0734%
1.5	0.3025%	0.0014%	1	25.8181%
1.75	0.7472%	0.0017%	1.25	56.3898%
2	1.1202%	0.0320%	1.5	84.2698%
2.5	0.7945%	0.1648%	—	—
3	0.7660%	0.8263%	—	—
3.5	0.8644%	0.5456%	—	—

Table 3: Standard deviation of failed points ratio per image by distance in meters.

d	D415	L515	d	SR305
0.5	0.0248%	0.0023%	0.3	0.0397%
0.75	0.0224%	0.0003%	0.4	0.0198%
1	0.0195%	0.0050%	0.5	0.0415%
1.25	0.0211%	0.0004%	0.75	0.0783%
1.5	0.0329%	0.0033%	1	0.1282%
1.75	0.0458%	0.0025%	1.25	0.1886%
2	0.0319%	0.0100%	1.5	0.2714%
2.5	0.0335%	0.0278%	—	—
3	0.0378%	0.0570%	—	—
3.5	0.0585%	0.0155%	—	—

Table 4: Average of outliers $\pm 10cm$ ratio per image by distance in meters.

d	D415	L515	d	SR305
0.5	0.1141%	0.0968%	0.3	0.1560%
0.75	0.1112%	0.0753%	0.4	0.0933%
1	0.0667%	0.0968%	0.5	0%
1.25	0.2809%	0.0968%	0.75	0.0081%
1.5	0.5387%	0.0968%	1	3.9062%
1.75	0.2568%	0.0813%	1.25	0.0032%
2	0.0558%	0.0957%	1.5	0.0063%
2.5	0.8361%	0.0882%	—	—
3	3.9436%	0.0839%	—	—
3.5	10.3307%	0.0609%	—	—

Table 5: Standard deviation of outliers $\pm 10cm$ ratio per image by distance in meters.

d	D415	L515	d	SR305
0.5	0.0209%	0%	0.3	0.0004%
0.75	0.0193%	0.0034%	0.4	0.0018%
1	0.0139%	1.2715%	0.5	0%
1.25	0.0226%	0%	0.75	0.0105%
1.5	0.0380%	0%	1	0.0003%
1.75	0.0431%	0.0025%	1.25	0.0046%
2	0.0168%	0.0008%	1.5	0.0057%
2.5	0.1892%	0.0018%	—	—
3	0.5601%	0.0027%	—	—
3.5	0.7226%	0.0001%	—	—

Table 6: Sensors' accuracy in terms of average root mean square point-to-plane distance error per image.

Cam_{dist}	$\pm 10cm$	$\pm 35cm$	All Points
$D415_{0,5m}$	0,002	0,004	0,122
$D415_{0,75m}$	0,003	0,003	0,087
$D415_{1m}$	0,005	0,005	0,079
$D415_{1,25m}$	0,007	0,007	0,037
$D415_{1,5m}$	0,009	0,009	0,139
$D415_{1,75m}$	0,013	0,013	0,293
$D415_{2m}$	0,017	0,017	0,093
$D415_{2,5m}$	0,025	0,027	0,036
$D415_{3m}$	0,036	0,042	0,069
$D415_{3,5m}$	0,045	0,063	0,065
$L515_{0,5m}$	0,001	0,001	134,853
$L515_{0,75m}$	0,001	0,001	202,285
$L515_{1m}$	0,001	0,001	150,913
$L515_{1,25m}$	0,002	0,002	7,556
$L515_{1,5m}$	0,002	0,002	14,588
$L515_{1,75m}$	0,002	0,002	22,389
$L515_{2m}$	0,002	0,002	15,542
$L515_{2,5m}$	0,003	0,003	14,865
$L515_{3m}$	0,004	0,004	25,025
$L515_{3,5m}$	0,006	0,006	13,835
$SR305_{0,3m}$	0,003	0,036	113,994
$SR305_{0,4m}$	0,005	0,005	114,889
$SR305_{0,5m}$	0,008	0,008	0,008
$SR305_{0,75m}$	0,009	0,009	0,009
$SR305_{1m}$	0,005	0,005	0,006
$SR305_{1,25m}$	0,007	0,007	0,008
$SR305_{1,5m}$	0,008	0,008	0,011

Table 7: Camera precision in terms of plane modelling consistency.

Cam_{dist}	$\overline{n_x}$	σ_{n_x}	$\overline{n_y}$	σ_{n_y}	$\overline{n_z}$	σ_{n_z}	$\overline{d}$	σ_d
$D415_{0,5m}$	3,9E-03	2,5E-04	4,1E-03	2,1E-04	1,0E+00	1,5E-06	-5,0E-01	1,1E-04
$D415_{0,75m}$	3,7E-03	3,0E-04	-8,8E-03	2,0E-04	1,0E+00	1,5E-06	-7,5E-01	2,2E-04
$D415_{1m}$	1,7E-02	7,2E-04	-1,3E-03	2,6E-04	1,0E+00	1,3E-05	-1,0E+00	1,6E-04
$D415_{1,25m}$	1,4E-02	1,6E-03	-2,7E-03	3,6E-04	1,0E+00	1,8E-05	-1,3E+00	3,1E-04
$D415_{1,5m}$	1,8E-02	8,7E-04	3,0E-03	4,3E-04	1,0E+00	1,6E-05	-1,5E+00	4,6E-04
$D415_{1,75m}$	2,2E-02	8,5E-04	3,4E-04	5,8E-04	1,0E+00	1,8E-05	-1,8E+00	4,9E-04
$D415_{2m}$	1,3E-02	2,9E-03	2,4E-03	8,6E-04	1,0E+00	2,9E-05	-2,0E+00	1,1E-03
$D415_{2,5m}$	1,0E-02	1,5E-03	-3,2E-03	9,3E-04	1,0E+00	1,4E-05	-2,5E+00	2,8E-03
$D415_{3m}$	1,4E-02	3,2E-03	-5,8E-04	1,1E-03	1,0E+00	3,3E-05	-3,0E+00	1,8E-03
$D415_{3,5m}$	6,6E-03	1,8E-03	1,8E-03	1,3E-03	1,0E+00	1,2E-05	-3,5E+00	3,2E-03
$L515_{0,5m}$	-4,6E-05	6,1E-04	-2,9E-03	2,3E-04	1,0E+00	7,9E-07	-5,0E-01	3,5E-04
$L515_{0,75m}$	1,8E-03	4,6E-04	-2,7E-03	1,6E-04	1,0E+00	7,1E-07	-7,5E-01	3,3E-04
$L515_{1m}$	-1,9E-03	6,1E-04	-5,7E-03	1,5E-04	1,0E+00	1,9E-06	-1,0E+00	3,5E-04
$L515_{1,25m}$	-6,8E-06	3,1E-04	2,1E-03	7,6E-05	1,0E+00	1,9E-07	-1,3E+00	3,1E-04
$L515_{1,5m}$	-3,1E-03	5,2E-04	-9,2E-04	1,9E-04	1,0E+00	1,7E-06	-1,5E+00	3,3E-04
$L515_{1,75m}$	1,8E-03	3,3E-04	3,1E-04	1,0E-04	1,0E+00	6,2E-07	-1,8E+00	3,1E-04
$L515_{2m}$	-8,0E-04	4,5E-04	-5,1E-04	1,7E-04	1,0E+00	4,4E-07	-2,0E+00	3,9E-04
$L515_{2,5m}$	3,2E-04	3,5E-04	-8,7E-04	1,4E-04	1,0E+00	1,7E-07	-2,5E+00	3,5E-04
$L515_{3m}$	-2,1E-03	3,3E-04	-3,9E-03	1,5E-04	1,0E+00	1,1E-06	-3,0E+00	3,3E-04
$L515_{3,5m}$	1,7E-03	4,0E-04	-6,8E-03	8,3E-05	1,0E+00	7,7E-07	-3,5E+00	3,5E-04
$SR305_{0,3m}$	4,4E-03	4,6E-03	-5,6E-03	1,0E-04	1,0E+00	2,7E-05	-3,0E-01	1,8E-04
$SR305_{0,4m}$	7,7E-03	5,2E-03	-8,2E-04	8,7E-05	1,0E+00	6,2E-05	-4,0E-01	2,3E-04
$SR305_{0,5m}$	1,7E-02	7,8E-03	5,2E-03	1,2E-04	1,0E+00	1,6E-04	-5,0E-01	2,7E-04
$SR305_{0,75m}$	-1,7E-02	1,3E-02	-1,3E-04	2,8E-04	1,0E+00	2,6E-04	-7,5E-01	6,3E-04
$SR305_{1m}$	-1,7E-02	7,3E-03	6,6E-03	1,8E-04	1,0E+00	1,0E-04	-1,0E+00	7,7E-04
$SR305_{1,25m}$	-2,7E-02	1,2E-02	-2,3E-03	5,0E-04	1,0E+00	2,6E-04	-1,3E+00	1,3E-03
$SR305_{1,5m}$	-2,3E-02	2,0E-02	-2,9E-03	1,1E-03	1,0E+00	5,8E-04	-1,5E+00	2,8E-03

Table 8: Camera precision in terms of plane normal vector angles standard deviation and spherical variance.

Cam_{dist}	$\overline{\phi}$	σ_ϕ	V	Cam_{dist}	$\overline{\phi}$	σ_ϕ	V
$D415_{0,5m}$	9,0E+01	1,5E-02	5,3E-08	$L515_{1,5m}$	9,0E+01	3,1E-02	1,5E-07
$D415_{0,75m}$	8,9E+01	9,2E-03	6,5E-08	$L515_{1,75m}$	9,0E+01	1,9E-02	5,9E-08
$D415_{1m}$	8,9E+01	4,1E-02	2,9E-07	$L515_{2m}$	9,0E+01	2,4E-02	1,2E-07
$D415_{1,25m}$	8,9E+01	8,5E-02	1,3E-06	$L515_{2,5m}$	9,0E+01	8,6E-03	7,3E-08
$D415_{1,5m}$	8,9E+01	4,9E-02	4,7E-07	$L515_{3m}$	9,0E+01	1,4E-02	6,5E-08
$D415_{1,75m}$	8,9E+01	4,9E-02	5,3E-07	$L515_{3,5m}$	9,0E+01	6,2E-03	8,3E-08
$D415_{2m}$	8,9E+01	1,6E-01	4,6E-06	$SR305_{0,3m}$	9,0E+01	1,5E-01	1,1E-05
$D415_{2,5m}$	8,9E+01	7,9E-02	1,6E-06	$SR305_{0,4m}$	9,0E+01	3,0E-01	1,4E-05
$D415_{3m}$	8,9E+01	1,7E-01	5,6E-06	$SR305_{0,5m}$	8,9E+01	4,2E-01	3,1E-05
$D415_{3,5m}$	9,0E+01	1,0E-01	2,5E-06	$SR305_{0,75m}$	8,9E+01	7,0E-01	9,0E-05
$L515_{0,5m}$	9,0E+01	1,4E-02	2,1E-07	$SR305_{1m}$	8,9E+01	3,2E-01	2,7E-05
$L515_{0,75m}$	9,0E+01	1,2E-02	1,2E-07	$SR305_{1,25m}$	8,8E+01	5,6E-01	7,4E-05
$L515_{1m}$	9,0E+01	1,8E-02	2,0E-07	$SR305_{1,5m}$	8,9E+01	1,0E+00	2,1E-04
$L515_{1,25m}$	9,0E+01	4,8E-03	5,2E-08	—	—	—	—