Object Detection and 6D Pose Estimation of Scenes with Occluded Objects Using Multiple Viewpoints

Development of a tool for annotation of image data, and an algorithm for visual inventory of supermarket items

Master's thesis in Complex Adaptive Systems

OSCAR BARK
ANDREAS GRIGORIADIS

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Object Detection and 6D Pose Estimation of Scenes with Occluded Objects Using Multiple Viewpoints

Development of a tool for annotation of image data, and an algorithm for visual inventory of supermarket items

OSCAR BARK
ANDREAS GRIGORIADIS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Computer vision and medical image analysis group
Chalmers University of Technology
Gothenburg, Sweden 2019

Object Detection and 6D Pose Estimation of Scenes with Occluded Objects Using Multiple Viewpoints
Development of a tool for annotation of image data, and an algorithm for visual inventory of supermarket items
OSCAR BARK
ANDREAS GRIGORIADIS

Supervisor: Lucas Brynte, Department of Electrical Engineering
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Computer vision and medical image analysis group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Object Detection and 6D Pose Estimation of Scenes with Occluded Objects Using Multiple Viewpoints
Development of a tool for annotation of image data, and an algorithm for visual inventory of supermarket objects

OSCAR BARK
ANDREAS GRIGORIADIS
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In recent years, deep learning based approaches have excelled at the task of object detection and 6D pose estimation at the expense of requiring large amounts of annotated data compared to traditional approaches. The purpose of this thesis is twofold. It investigates an algorithm for object detection and 6D pose estimation with the aim of enabling visual inventory of items in a supermarket, and it presents the development of a tool for annotation of datasets suitable for such tasks. The detection algorithm aimed to exploit images from multiple viewpoints of the same scene to perform the task in scenarios where established methods of single-view detection are unsuitable. A state-of-the-art single-view pose estimation method was built upon to detect multiple instances of an object, and was used on each frame of a video of the scene. The output from each frame was then aggregated by clustering candidate pose centers, enabling grouping of keypoint locations across images to which the final detections of poses could be fitted. The annotation tool relied on users identifying keypoint locations in a few images, and exploited a known camera motion to infer the poses of objects relative all viewpoints. The detection algorithm was evaluated on three previously unseen scenes with a total of 85 objects, and yielded F_1-scores in the range of 0.1739 to 0.7142 for the different classes, implying that the algorithm was not suitable for the task. We attribute the low performance partly to our method of grouping keypoints to separate instances not being able to handle scenarios where objects are occluded, and partly to learning and generalization issues of the deep learning module. Conversely, we consider the annotation tool successful, as it integrates several features to facilitate the annotation, and the method is robust. However, users need to be cautious during annotation due to potential inaccuracies in the camera model assumed, and the difficulty of labeling rotationally symmetric objects appropriately. A new dataset exhibiting difficulties present in the target setting was created, and annotated using our tool.

Keywords: Object Detection, 6D Pose Estimation, Data Annotation, Visual Inventory, Deep Learning

Acknowledgements

We would like to thank our examiner Fredrik Kahl and our supervisor Lucas Brynte for all the great support and motivation during our work. We are grateful to Fredrik for the inspiration during the computer vision course that motivated us to pursue this thesis, for excellent guidance in both general and technical matters, and for help with the formulation and development of the thesis. To Lucas we are grateful for the numerous inspiring technical meetings, the support provided during times of adversity and the excellent knowledge he possesses on the subject. We would also like to express gratitude to Sebastian Almfeldt for helping us produce essential models for our objects.

Oscar Bark & Andreas Grigoriadis, Gothenburg, August 2019

Glossary

Pinhole camera model: Simple model of a camera, describing how points in a 3D world are displayed in a 2D picture.
6D Pose: A representation of an objects location and orientation. 6D indicates six degrees of freedom.
Motion: A collection of poses for a set of cameras.
Keypoint: Point with special significance, e.g. on a point cloud model of an object.
Clustering: The task of partitioning a set of data into groups.
MSC: Mean-Shift Clustering. A clustering method which can determine the number of groups.
Mode: The most likely value of a probability distribution.
Model fitting: Adjusting the parameters of a model to make it coherent with data.
RANSAC: Random sample consensus. Method to fit a model to data with outliers, by sampling of small subsets.
ANN/NN: Artificial Neural Network
CNN: Convolutional Neural Network, a variant of artificial neural networks often used for modeling functions with images as input.
Deep Learning: Usage of large ANN's to solve complex tasks.
Dataset: A set of labeled data related to a task.
F_1**-score**: A performance metric for classification tasks which is especially suitable for unbalanced datasets.
Generalizability: Ability of an algorithm to perform well on previously unseen data.
Hyperparameters: Algorithm parameters which are not automatically optimized with respect to data.
Overfitting: Fitting a model to a specific observation of data, resulting in poor generalizability of the model.
PyTorch: A programming library for Python for implementing deep learning models

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Background . . . 1
1.2 Purpose . . . 2
1.3 Specification of issues under investigation . . . 3
1.4 Limitations . . . 3
1.5 Related work . . . 4
1.5.1 Annotation of 6D poses . . . 4
1.5.2 Single-view 6D pose estimation . . . 5
1.5.3 Multi-view 6D pose estimation . . . 5
1.6 Contributions . . . 6
1.7 Report disposition . . . 6

2 Theory **9**
2.1 Geometric computer vision . . . 9
2.1.1 Pinhole camera model . . . 9
2.1.2 Inference from camera equation . . . 12
2.1.3 Pose representation . . . 14
2.1.4 Pose optimization with plane constraint . . . 17
2.1.5 Inferring the image projection . . . 18
2.1.6 Object model representation and keypoints . . . 18
2.2 Deep learning . . . 19
2.2.1 Convolutional neural networks . . . 21
2.3 Object detection and 6D pose estimation . . . 22
2.3.1 Evaluation . . . 23
2.4 Clustering algorithms . . . 24
2.4.1 K-Means clustering . . . 25
2.4.2 Mean-shift clustering . . . 25

3 Project disposition & dataset acquisition **27**
3.1 Project planning and execution . . . 28
3.2 Dataset creation . . . 28
3.2.1 Resulting dataset . . . 29

4 Annotation Tool **33**
4.1 Core idea and specification . . . 33
4.2 Method . . . 34
4.2.1 Software development framework . . . 34
4.2.2 Underlying mechanisms . . . 34
4.2.3 Experiment 1: Assessing quality of labels . . . 36
4.2.4 Experiment 2: Assessing the efficiency of annotation . . . 37
4.3 Resulting software . . . 37
4.3.1 Main view . . . 37
4.3.2 Point identification . . . 38
4.3.3 Manual pose manipulation . . . 40
4.3.4 Plane identification . . . 41
4.4 Experiment results . . . 42
4.5 Discussion . . . 44
4.5.1 Summary of key features . . . 44
4.5.2 Workload analysis . . . 44
4.5.3 Visual assessment of pose label quality . . . 45
4.5.4 Flaws of the tool . . . 46
4.5.5 Future work . . . 47

5 Detection Algorithm **49**
5.1 Method . . . 49
5.1.1 Single-view pose estimation using PVNet . . . 51
5.1.2 Our single-view pipeline . . . 55
5.1.3 Aggregation . . . 60
5.2 Results . . . 62
5.3 Discussion . . . 64
5.3.1 Design choices and potential improvements . . . 64

6 Conclusions **67**

References **69**

A Voting field prediction for tightly packed objects **I**

B Dataset overview **III**

C Samples of annotation segmentations **VII**

List of Figures

2.1 Pinhole camera model 10
2.2 Schematics of a feed-forward artificial neural network. 20
2.3 Discrete convolution operation 22

3.1 Flowchart diagram of project tasks. 27
3.2 Two of the scenes in the dataset 30

4.1 Main view after loading data of a scene in the Annotation Tool . . . 38
4.2 View where user may identify 2D keypoint locations in the Annotation Tool 39
4.3 Prompt view to manually select one of two initial pose candidates in the Annotation Tool 39
4.4 Fine adjustment view in the Annotation Tool 41
4.5 Histogram of the number of 2D points identified for fitting the pose of an object instance in the Annotation Tool 43
4.6 Histogram of the number of viewpoints utilized for fitting the pose of an object instance in the Annotation Tool. 43

5.1 Overview of the Detection Algorithm 50
5.2 Example of a voting field corresponding to a keypoint. 53
5.3 Example of semantic segmentation 53
5.4 Flowchart of single-view meachinsm in the Detection Algorithm . . . 57
5.5 Voting direction vs. principal component of data. 59
5.6 Voting directions for a sparse set of keypoint hypotheses. 59
5.7 Point estimates from single-view procedure for two nearby viewpoints. 61

A.1 Comparison between ground truth and predicted voting field for a keypoint in an image with tightly placed objects I

B.1 Overview of all scenes in the dataset V

C.1 Overview of produced pose labels VIII

List of Tables

3.1 Statistics of training dataset. The final 7 columns are the number of instances of each class as indexed in Table 3.3. 30
3.2 Statistics of validation dataset. The final 7 columns are the number of instances of each class as indexed in Table 3.3. 31
3.3 Table of classes used in our dataset, and their corresponding numeric indices for future reference. 31

5.1 F_1-score over all scenes in the validation set for each class. 62
5.2 F_1-score over all classes for each scene in the validation set. 62
5.3 F_1-score over all scenes in the training set for each class. 63
5.4 F_1-score over all classes for each scene in the training set. 63
5.5 Table of segmentation loss, voting loss and total loss values on training and validation data for the different classes. The classes are indexed as in Table 3.3. 64

1

Introduction

This chapter serves to present the context of this project, and to specify the main problems it aims to solve. Furthermore, the disposition of the report is presented.

1.1 Background

In May of 2016, the research project *Semantic Mapping and Visual Navigation for Smart Robots* was initiated at Chalmers University of Technology with the goal of developing an integrated framework for autonomous vehicles to see, navigate in, and map their surroundings based on computer vision and optimal control techniques [1]. The project owners aim to demonstrate their advances in relevant fields by developing an autonomous system capable of performing visual inventory inspection in a supermarket setting, using small quadcopters. An important problem that needs to be solved for this task is that of identifying different classes of stocked items and their locations in shelves using camera images.

A common problem within the field of robotics is object detection and 6D pose estimation from single images of scenes containing various items. Such solutions often utilize deep learning models due to their robustness to lighting conditions, and their ability to exploit semantic information [2]. The models often make up part of the application pipeline in conjunction with conventional computer vision methods, but more recently researchers have started to design end-to-end systems where all operations are learned from data [3]. Being able to identify visible objects and their poses in real time from a single image is important for a robot which aims to navigate a dynamic world.

In some settings however, the aim is simply to obtain a complete map of a static scene, containing all objects and their poses in a global coordinate system. This is the case with our problem of visual inventory posed above, and here there are no constraints on real-time performance or number of viewpoints. A challenge posed when applying single-view estimation methods to solve this problem is that objects in a scene may be partially or wholly occluded in images. Though detection and pose estimation of partially occluded objects have been successfully achieved with single-view methods in settings with few selected objects [4], in many practical settings,

such as supermarket shelves, it is not feasible to produce a single image where all items are sufficiently visible. In this light, the idea of exploiting multiple viewpoints of the scene to overcome said challenges has been explored [5] [6] [7], albeit to a lesser extent than single-view methods.

Furthermore, a challenge of using deep learning frameworks is the need for large amounts of annotated data on which to train the models. Producing this data manually can be difficult and time consuming for a party who wishes to apply these methods in new settings. Typically, datasets for these tasks are produced by constructing specific scenes including clear markers from which the poses of carefully placed objects can be calculated [2] [8], or by manually rotating and translating 3D models in a point cloud reconstruction of the scene at hand [9]. It is common to generate synthetic training data, e.g. images obtained by rendering CAD-models of relevant objects against a black background [10] [11]. This is due to convenience, and lack of real data [12]. Models trained on synthetic data risk not being able to generalize well to real data without careful strategies to combat overfitting.

1.2 Purpose

In light of the problems presented above, the purpose of this project was twofold:

- It aimed to develop and evaluate an algorithm which uses a video stream of a scene for global detection and 6D pose estimation of objects which are occluded in several of the frames. The method is primarily meant to be applied for inventory of supermarket products placed in shelves, as to contribute to the above described research project.

- It aimed to develop and evaluate a method for semi-automatic annotation of data used for object detection and pose estimation. The produced data should be compatible with algorithms such as the above mentioned single-view pose estimation methods. The intention was to provide an easy-to-use tool for researchers to create large scale annotated data sets.

The algorithm for object detection and 6D pose estimation will in this report be referred to as the *Detection Algorithm*, whereas the method for data annotation will be referred to as the *Annotation Tool*. Furthermore, a new dataset was created for this project, which we annotated using the Annotation Tool, and on which we evaluated the Detection Algorithm.

1.3 Specification of issues under investigation

Concerning the Annotation Tool we aimed to develop, the following questions were posed:

- Does our annotation tool facilitate the creation of large datasets for object detection and pose estimation in new settings?
- How accurate are labels obtained from typical use cases with our annotation tool?

Concerning the Detection Algorithm, the following questions were posed:

- Is it possible to accurately identify all objects in an occluded scene, with reasonable pose estimates, using only RGB images from multiple viewpoints?
- How can images of a static scene from several viewpoints be exploited to enable object detection and pose estimation with a deep learning framework?
- How do different levels of occlusion of objects in a scene affect the performance of visual inventory with multiple images?

1.4 Limitations

Each entry in the dataset we produced consists of an RGB video of a scene with its corresponding *motion*, and the poses of each object in the scene with respect to every camera. The dataset does not contain labels for other relevant quantities such as segmentation labels, object visibility in the images, etc.

Methods for obtaining the motion of the video, which is an important quantity in many multi-view computer vision applications (see Section 2.1), were not investigated for this project. The motion was instead obtained by using an openly licensed Structure-from-motion software solution which uses only the video to calculate it. The software used was COLMAP [13] [14], which imposes constraints on the videos in order to function properly. Such constraints include good textures and similar illumination.

Although many learning based computer vision applications make use of labels other than poses, the Annotation Tool only produces the pose labels. Relevant labels, such as segmentations, keypoint projections and 2D bounding boxes may be inferred in an automated fashion using the produced pose labels (and we did use these labels in our project). The motivation for not integrating such features into the tool was twofold: Separate tools implementing such features already exist and it would have required additional development time. The tool focuses on minimizing the manual effort required to produce a set of principal labels which can then be used to produce

additional labels.

As was previously mentioned, the project *Semantic Mapping and Visual Navigation for Smart Robots* strives to perform full visual inventory of market shelves using drones. Since our algorithm aimed to be used for this type of task, the evaluation of our work did not focus on testing for high accuracy of the poses of the objects. Rather, the evaluation was based on the detection rate, i.e detecting a majority of the objects while avoiding false detections.

There is conceptually no need for the visual inventory task to be performed live during drone deployment. Therefore, neither does the algorithm need to process data sequentially, nor did we impose real-time performance constraints on the execution speed.

Furthermore, the inventory algorithm was developed with aim of being applied in a supermarket setting, but limitations regarding this goal had to be made. Firstly, the number objects used during development and testing was for performance and convenience limited to a manageable number of objects, rather than the number of objects a typical store could have in stock, ranging in the tens of thousands. Secondly, supermarket shelves may also vary greatly in shape, causing various degrees of unpredictable environmental occlusions (see Section 3.2). The scope of applicable cases for the project had to be limited by making assumptions regarding the environment which may or may not be valid for the supermarket setting. Thirdly, the objects were limited to rigid objects, since a static scene is assumed in the scope of this project. Consequently, we did not expect the final algorithm, regardless of results on our data, to be able to perform the task in real typical supermarkets where instances of products such as fruit vary in shape.

1.5 Related work

In this section, related works on tasks similar to ours are reviewed.

1.5.1 Annotation of 6D poses

As previously mentioned, existing methods for 6D pose annotation include careful placement of an object on a board with markers [2] [8] and manual fitting of an object model to a point cloud [9]. In the former case, the requirement set on the pose placement and the use of the board makes such a method unfeasible in a setting where no constraints are to be set on the environment. For the latter case, the method is limited to scenes where the object is clearly visible in many images. This is not the case when objects are occluded, and sets a restriction on the environment.

1.5.2 Single-view 6D pose estimation

Traditionally, multiple methods for 6D pose estimation depended on template matching techniques. For the 6D pose problem this usually means rendering object models to an image and using a handcrafted template to compute a similarity score at each image location [15] [16]. These methods are however sensitive to changes in appearance and occlusions in the environment.

Another category of pose estimation methods are feature based methods which aim to find local features of objects [17] [18] and match them between the image and an object model. Such procedures yield 2D-3D correspondences which can be used to solve the Perspective-n-Point (PnP) problem to find the pose. Traditional methods within this category make use of handcrafted features and the approach is efficient for pose estimation even in the case of occlusion. The methods do however suffer performance losses when an object lacks texture, or the image is blurry.

Recent years have shown a trend towards deep learning based methods, in particular Convolutional Neural Networks, for learning features [19]. Breaking down the problem into the steps of keypoints detection, by utilization of a CNN, and solving the PnP has been proven effective in multiple works. The work of [4] performs the three steps of instance segmentation, pixel-wise regression of 3D coordinates for 2D-3D correspondences and solving the PnP. A pipeline that regresses pixel-wise voting for the direction to 2D keypoints, and utilized an uncertainty measure for solving an alternate formulation of the PnP was proposed by [20]. This work currently achieves state-of-the-art for numerous standard datasets [20] in the 6D pose localization task.

Other works propose to utilize a CNN to solve the problem in an end-to-end fashion by directly regressing the pose parameters or a subset of them. A completely end-to-end system was designed by [3] where a VGG network is combined with a region proposal network that simultaneously classifies and regresses the class, bounding box, mask and pose of an object. [10] proposes to regress voting for each pixel in a instance segmented area towards the center 2D point of the object, and the neural network also estimates the depth of this point. The rotational part is also regressed by the network.

1.5.3 Multi-view 6D pose estimation

To detect global poses given multiple viewpoints, multiple works have approached the problem by using a single view pose algorithm and aggregation of the output. The single view pose algorithm of [3] is extended by [6] to the multi view case. Pose hypotheses for each image are estimated by using a sparse auto-encoder network with RGB-D input, followed by a Hough forest classifier. The pose hypotheses from all images are transformed to a global coordinate system where the final poses are determined by subtractive clustering [21] of the pose centers. They also handle the case of multiple instances of the same class by the hypothesis verification method

proposed by [22]. The work of [23] integrates the output from several single-view pose algorithms, that each provide pose hypotheses coupled with a confidence score. They predict poses by weighted Mean-Shift clustering in the state-space of the pose. [5] uses semantic segmentation in conjunction with a created point cloud of the scene by exploiting RGB-D data. The segmentation output from a CNN is projected to the corresponding 3D points which yields a point cloud of 3D semantic points. The point cloud of the object is adjusted to fit to the semantic 3D points by the Iterative closest point algorithm.

1.6 Contributions

The main contributions of this thesis work are the following:

- A new dataset for pose estimation in scenes with heavy occlusions between multiple instances of the same objects.
- A tool for 6D pose annotation of multiple RGB images capturing a general environment.
- A proposed framework for global object detection and pose estimation using a sequence of RGB images, and an evaluation of said framework.

1.7 Report disposition

In this section, the contents of the remaining chapters are briefly introduced.

The Theory chapter aims to present concepts within computer vision that are related to our work. Understanding the concepts presented in this chapter is important to understand our solutions. The chapter presents all theory, both that which is related to the Annotation tool, and that which is related to the Detection Algorithm.

The Project Disposition & Data Acquisition chapter presents the disposition of the project as a whole. It covers practical details such as choice of platforms and tools used in the development of the solutions, details regarding the data acquisition and the overarching structure of the solutions.

This project, as previously described, has two main purposes for which the tasks differ substantially. Therefore, method, results and discussion of the two sub-projects are for clarity's sake reported in two separate chapters: Annotation Tool and Detection Algorithm.

Finally, in the conclusion chapter, the key takeaways from the development of the Annotation Tool, and the resulting Detection Algorithm, are summarized and com-

mented on.

2

Theory

2.1 Geometric computer vision

The general field of Computer Vision deals with extraction of data from images and includes several categories of problems. One particular field concerns modeling the world as 3D points and how these are captured in an image in terms of geometrical 2D points. In these cases, linear algebra can be utilized for predicting e.g. the position and shape of objects, the movement pattern of a moving camera and the depth map in an image. Multiple models exist in this category and in this section one of the most widely used is presented.

2.1.1 Pinhole camera model

The *Pinhole camera model* models how a camera captures a 3D point in the world on the sensor/film by projecting the 3D points onto a geometrical plane called the *image plane*, resulting in 2D points. The 3D and 2D points are denoted *object points* and *image points* respectively. The projection of a single object point $\boldsymbol{X}$ can be viewed in Figure 2.1.

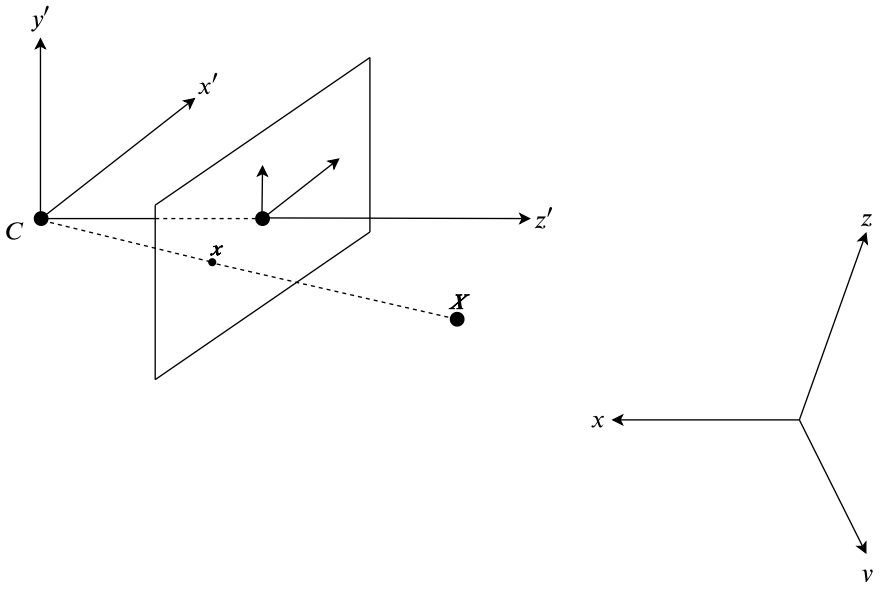


Figure 2.1: Illustration of the pinhole camera model. A world coordinate $\boldsymbol{X}$ is projected onto the image plane of distance 1 length unit from the camera center $\boldsymbol{C}$, resulting in the image point $\boldsymbol{x}$. The camera is located in a coordinate system (x, y, z) and has its own coordinate system (x', y', z').

As the figure depicts, a camera is located at a point $\boldsymbol{C}$. The orientation of the camera is defined by a *camera coordinate system* where the Z-axis, denoted the *principal axis*, serves as the viewing direction of the camera. The camera film is modeled by a plane orthogonal to the principal axis at distance 1 from $\boldsymbol{C}$. The projection of an object point onto the image plane is done by taking the intersection of the line defined by $\boldsymbol{X} - \boldsymbol{C}$ and the image plane, resulting in a coordinate $\begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$, where $\boldsymbol{x} \in \mathbb{R}^2$ is the image point.

The camera coordinate system can be viewed as the local frame of reference of the world as "seen" by the camera. The camera, along with the object points, may however be described in an outer *global coordinate system* which is more descriptive of the world itself and is arbitrarily defined. As such, the position of $\boldsymbol{C}$ denotes the position of the camera in the global coordinate system while this point is defined to be the origin of the camera coordinate system. See Figure 2.1

When dealing with the projections of object points into cameras in different coordinate systems, transformations need to be utilized. A Euclidian transformation of an object point $\boldsymbol{X}$ from a global to a camera coordinate system is composite of a rotational and translation operation which is performed with matrix operations

$$\boldsymbol{X}' = R\boldsymbol{X} + \boldsymbol{t} \tag{2.1}$$

where $\boldsymbol{X}' = \begin{pmatrix} X'_1 \\ X'_2 \\ X'_3 \end{pmatrix}$ is the object point in the camera coordinate system, R is a

rotational matrix (see Section 2.1.3.2) and $\boldsymbol{t}$ is a vector for translation. The projected image point of $\boldsymbol{X}'$ onto the image plane is given by the *perspective projection function*

$$\begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = \pi(\boldsymbol{X}') = \frac{\boldsymbol{X}'}{X_3'} . \tag{2.2}$$

A more common way of writing the above equation is

$$\lambda \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = \boldsymbol{X}' \tag{2.3}$$

where $\lambda = X_3'$. Using the two equations (2.1) and (2.3) for transformation and projection yields the *normalized camera equation* (2.4) for capturing an object point, residing in a global coordinate system, in a camera.

$$\lambda \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = R\boldsymbol{X} + \boldsymbol{t} \tag{2.4}$$

In addition to the coordinate system transformation, a transformation needs to be made to account for the intrinsic properties of the camera, such as pixel resolution. This is done by a further multiplication by a matrix K which among other things changes units from spatial image distance, e.g. meters, to pixels. This matrix is added to (2.4) which gives us the *camera equation*

$$\lambda K \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = K(R\boldsymbol{X} + \boldsymbol{t}) = K \begin{bmatrix} R & \boldsymbol{t} \end{bmatrix} \begin{pmatrix} \boldsymbol{X} \\ 1 \end{pmatrix} .$$

The matrix $P = K \begin{bmatrix} R & \boldsymbol{t} \end{bmatrix}$ is called the *camera matrix*. K may be a known quantity, in which case the camera is said to be *calibrated*, and it is often eliminated from the problem. The *normalized camera matrix* may then be defined as $P = \begin{bmatrix} R & T \end{bmatrix}$.

The points $\begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$ and $\begin{pmatrix} \boldsymbol{X} \\ 1 \end{pmatrix}$ are expressed in *homogeneous coordinates* which are defined merely to express the equations in a compact and convenient way. Throughout the rest of the report, the 1 will most often be omitted for brevity and $\boldsymbol{X}$ and $\boldsymbol{x}$ will denote both the original and homogeneous versions of the points. A more compact way to write the (normalized) camera equation is then

$$\lambda \boldsymbol{x} = P\boldsymbol{X} = R\boldsymbol{X} + \boldsymbol{t}$$

which will also be denoted the camera equation throughout this report. Note that the camera equation is in fact composed of three scalar equations

$$\begin{cases} \lambda x = R_1^T \boldsymbol{X} + t_1 \\ \lambda y = R_2^T \boldsymbol{X} + t_2 \\ \lambda = R_3^T \boldsymbol{X} + t_3 \end{cases}$$

where R_i^T denotes the i:th row of R and $\boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$.

2.1.2 Inference from camera equation

The camera equation serves as a basis for inferring different quantities depending on the known data. In the general case the known data includes multiple points N which give rise to an equation system of the following form:

$$\lambda_i \boldsymbol{x}_i = P\boldsymbol{X}_i,\ i = 1, .., N\,. \tag{2.5}$$

The following problems are relevant for this work.

2.1.2.1 Structure from motion

When a set of N corresponding (belonging to the same object point) image points $\{x_i, \bar{\boldsymbol{x}}_i\}_{i=1}^N$ are known in two different images, the unknowns of the problem are the normalized camera pairs $P_1 = [R_1\ \boldsymbol{t}_1]$ and $P_2 = [R_2\ \boldsymbol{t}_2]$, the set of object points $\{\boldsymbol{X}_i\}_{i=1}^N$ and the set of point depths $\left\{\lambda_i, \bar{\lambda_i}\right\}_{i=1}^N$ in the following equation system:

$$\begin{cases} \lambda_i \boldsymbol{x}_i = P_1 \boldsymbol{X}_i \\ \bar{\lambda_i} \bar{\boldsymbol{x}}_i = P_2 \boldsymbol{X}_i,\ i = 1, .., N\,. \end{cases}$$

There is an ambiguity in this equation system. As the cameras and object points are unknown, a transformation H can be applied and solving an equivalent problem:

$$\begin{cases} \lambda_i \boldsymbol{x}_i = P_1 \boldsymbol{X}_i = (P_1 H)(H^{-1}\boldsymbol{X}_i) = \tilde{P_1}\tilde{\boldsymbol{X}}_i = \tilde{R_1}\tilde{\boldsymbol{X}}_i + \tilde{\boldsymbol{t}_1} \\ \bar{\lambda_i} \bar{\boldsymbol{x}}_i = P_2 \boldsymbol{X}_i = (P_2 H)(H^{-1}\boldsymbol{X}_i) = \tilde{P_2}\tilde{\boldsymbol{X}}_i = \tilde{R_2}\tilde{\boldsymbol{X}}_i + \tilde{\boldsymbol{t}_2}\,. \end{cases}$$

The transformation may be chosen to simplify the problem:

$$H = \begin{bmatrix} R_1^{-1} & -R_1^{-1}\boldsymbol{t}_1 \\ 0 & 1 \end{bmatrix} \implies \tag{2.6}$$

$$\tilde{P_1} = [R_1\ \boldsymbol{t}_1]H = [I\ 0] \tag{2.7}$$

$$\tilde{P_2} = [R_2\ \boldsymbol{t}_2]H = [R_2 R_1^{-1}\ -R_2 R_1^{-1}\boldsymbol{t}_1 + \boldsymbol{t}_2]\,. \tag{2.8}$$

As can be seen, the solution for one of the cameras is arbitrary and can be constructed to be the *identity camera* $P = \begin{bmatrix} I & 0 \end{bmatrix}$. The solutions of the cameras do in fact represent how they are oriented relative to each other. Note that there is no connection in this problem to any global coordinate system. As such, a global coordinate system may be defined to be aligned with say P_1, which is effectively what is being done when it is transformed to be the identity camera.

There is also another ambiguity in the equation system (2.1.2.1). Since only the image coordinates x_i are known for each camera, an equivalent equation system can be acquired by multiplying each equation by a scalar s

$$\begin{cases} \lambda_i s\boldsymbol{x}_i = sR_1\boldsymbol{X}_i + s\boldsymbol{t}_1 \\ \bar{\lambda}_i s\bar{\boldsymbol{x}}_i = sR_2\boldsymbol{X}_i + s\boldsymbol{t}_2 \end{cases} \implies$$

$$\begin{cases} \lambda'_i\boldsymbol{x}_i = R_1\boldsymbol{X}'_i + \boldsymbol{t}'_1 \\ \bar{\lambda}'_i\bar{\boldsymbol{x}}_i = R_2\boldsymbol{X}'_i + \boldsymbol{t}'_2 \end{cases}$$

which is called a *scale ambiguity*. Given no other information regarding the scenery, s will be a unknown parameter.

In a more general case, the structure from motion consist of image point correspondences in m images and the unknowns will include the cameras $\{P_1,\ P_2, ...,\ P_m\}$. Similar to the two-view case, a transformation H may be applied to align a global coordinate system with the first camera, as seen in eq. (2.10).

$$\{P_1H,\ P_2H, ...,\ P_mH\} = \left\{\begin{bmatrix} I & 0 \end{bmatrix},\ P'_2, ...,\ P'_m\right\} \tag{2.9}$$

This set of cameras is denoted a *motion object*. In the case that the scale s is unknown, the set of cameras is denoted an *unscaled motion object*

$$\left\{\begin{bmatrix} I & 0 \end{bmatrix},\ \begin{bmatrix} R'_2 & s\boldsymbol{t}'_2 \end{bmatrix}, ...,\ \begin{bmatrix} R'_m & s\boldsymbol{t}'_m \end{bmatrix}\right\}. \tag{2.10}$$

2.1.2.2 Perspective-n-point

When the image points in a single image and the object points are known, in which case they are said to be *2D-3D correspondences*, the unknowns of the camera equation are λ_i and P. This problem is called the *Perspective-n-point (PnP)* problem. Two variations of the problem exist

- The object points are defined in a given global coordinate system, in which case the solution to the problem will be the camera matrix.
- The object points are defined in a local coordinate system (e.g. for an object model, see 2.1.6) in which case the solution will be to find the position and orientation of the object relative to the camera.

In the latter case, the global coordinate system is undefined and the problem may be viewed in two equivalent ways. Either the global coordinate system can be defined to be aligned with the object local coordinate system and the problem is to find the camera matrix. Or it can be aligned with the camera coordinate system (similar to the operations in (2.6), resulting in the identity camera) and the problem is finding a transformation $T = \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix}$ from the object coordinate system to the global

coordinate system such that it projects to the image points in camera $P = \begin{bmatrix} I & 0 \end{bmatrix}$ according to eq. (2.11).

$$\lambda \boldsymbol{x} = PT\boldsymbol{X} = [I\ 0] \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} \boldsymbol{X} = [R_p\ \boldsymbol{t}_p]\boldsymbol{X} \tag{2.11}$$

As can be seen this equation has the same form as the camera equation, showing that the two ways of looking at the problem are equivalent.

When solving the equations that arise, numerical methods need to be resorted to since exact solutions rarely occur due to noise in the data. One of several ways of solving the PnP problem is minimizing the *reprojection error*

$$E = \sum_{k=1}^{K} \boldsymbol{r}_k^T \boldsymbol{r}_k \tag{2.12}$$

$$\boldsymbol{r}_k = \pi(R\boldsymbol{X}_k + \boldsymbol{t}) - \boldsymbol{x}_k \tag{2.13}$$

where π is the perspective projection function from (2.2), K is the total number of point correspondences and $\boldsymbol{r}_k$ is denoted a *residual vector*.

An extension to this problem is the case of having 2D-3D correspondences in multiple images of a motion object and finding a solution for T to the following equation system:

$$\begin{cases} \lambda \boldsymbol{x}_1 = P_1 T \boldsymbol{X} = [I\ 0] \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} \boldsymbol{X} \\ \lambda \boldsymbol{x}_2 = P_2 T \boldsymbol{X} = [R_2\ \boldsymbol{t}_2] \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} \boldsymbol{X} \\ ... \\ \lambda \boldsymbol{x}_m = P_m T \boldsymbol{X} = [R_3\ \boldsymbol{t}_3] \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} \boldsymbol{X}\,. \end{cases}$$

Solving this can be done by minimizing the *multi-view reprojection error*

$$E_{multi} = \sum_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}_i} \boldsymbol{r}_{i,k}^T \boldsymbol{r}_{i,k} \tag{2.14}$$

$$\boldsymbol{r}_{i,k} = \pi(R_i \boldsymbol{X}_k + \boldsymbol{t}_i) - \boldsymbol{x}_{i,k} \tag{2.15}$$

where $\mathcal{M}$ is the set of camera indices in the motion object, and $\mathcal{K}_i$ is the set of point correspondence indices available for camera i.

2.1.3 Pose representation

As was described by (2.11), the projection of an object point $\boldsymbol{X}$ in some local coordinate system onto a camera is performed by multiplication by a matrix $P_p =$

$[R_p \ \boldsymbol{t}_p]$. P_p is commonly referred to as the *pose* of an object and represent the location and orientation relative a particular camera. Note that $\boldsymbol{t}$ represents the placement of the local coordinate system origin in the global coordinate system. This can be seen by transforming the origin point

$$\boldsymbol{X}' = R_p \boldsymbol{X}_{origin} + \boldsymbol{t}_p = R_p \boldsymbol{0} + \boldsymbol{t}_p = \boldsymbol{t}_p \,. \tag{2.16}$$

In the case of representing a pose relative to a motion object, where it makes more sense to use a global coordinate system, a *global pose* is defined to the pose relative to the first camera. For the pose $P_p = [R_p \ \boldsymbol{t}_p]$ in a camera $P = [R \ \boldsymbol{t}]$ of the motion object, the pose $P'_p = [R'_p \ \boldsymbol{t}'_p]$ relative any another camera $P' = [R' \ \boldsymbol{t}']$ can be calculated by

$$P'_p = P' \begin{bmatrix} R^{-1} & -R^{-1}\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} . \tag{2.17}$$

2.1.3.1 Global pose with unscaled motion object

When the motion object does not share the scale units with the object point clouds, the pose transformation as given by equation (2.17) between cameras is invalid. Consider two cameras $\left\{ \begin{bmatrix} R_1 & s\boldsymbol{t}_1 \end{bmatrix}, \begin{bmatrix} R_2 & s\boldsymbol{t}_2 \end{bmatrix} \right\}$ of an unscaled (s unknown) motion object for which a global coordinate system is defined. For known image points $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and a given object model $\{\boldsymbol{X}_i\}_{i=1}^m$ the global pose has a transformation $T = \begin{bmatrix} R_T & \boldsymbol{t}_T \\ 0 & 1 \end{bmatrix}$ to the global coordinate system which needs to fulfill the equations

$$\lambda_1 \boldsymbol{x}_1 = P_1 T \boldsymbol{X} = \begin{bmatrix} R_1 & s\boldsymbol{t}_1 \end{bmatrix} \begin{bmatrix} R_T & \boldsymbol{t}_T \\ 0 & 1 \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} R_1 R_T & R_1 \boldsymbol{t}_T + s\boldsymbol{t}_1 \end{bmatrix} \boldsymbol{X} \tag{2.18}$$

$$\lambda_2 \boldsymbol{x}_2 = P_2 T \boldsymbol{X} = \begin{bmatrix} R_2 & s\boldsymbol{t}_2 \end{bmatrix} \begin{bmatrix} R_T & \boldsymbol{t}_T \\ 0 & 1 \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} R_2 R_T & R_2 \boldsymbol{t}_T + s\boldsymbol{t}_2 \end{bmatrix} \boldsymbol{X} \,. \tag{2.19}$$

The two equations can be solved individually for $P_{sol1} = P_1 T$ and $P_{sol2} = P_2 T$. The translational part of the solutions $\boldsymbol{t}_{sol1}$ and $\boldsymbol{t}_{sol2}$ can be used to determine the value for s by

$$\boldsymbol{t}_{sol1} = R_1 \boldsymbol{t}_T + s\boldsymbol{t}_1 \tag{2.20}$$

$$\boldsymbol{t}_{sol2} = R_2 \boldsymbol{t}_T + s\boldsymbol{t}_2 \implies \tag{2.21}$$

$$R_1^T \boldsymbol{t}_{sol1} = \boldsymbol{t}_T + s R_1^T \boldsymbol{t}_1 \tag{2.22}$$

$$R_2^T \boldsymbol{t}_{sol2} = \boldsymbol{t}_T + s R_2^T \boldsymbol{t}_2 \implies \tag{2.23}$$

$$(R_1^T \boldsymbol{t}_{sol1} - R_2^T \boldsymbol{t}_{sol2}) = s(R_1^T \boldsymbol{t}_1 - R_2^T \boldsymbol{t}_2) \implies \tag{2.24}$$

$$\frac{(R_1^T \boldsymbol{t}_1 - R_2^T \boldsymbol{t}_2)^T (R_1^T \boldsymbol{t}_{sol1} - R_2^T \boldsymbol{t}_{sol2})}{||(R_1^T \boldsymbol{t}_1 - R_2^T \boldsymbol{t}_2)||_2^2} = s \,. \tag{2.25}$$

This provides a way to determine s given an unscaled motion object, image points and model points. It also shows that it is necessary to determine s before the pose transformation (2.17) is used.

2.1.3.2 Rotation representation

A rotation matrix $R \in \mathbb{R}^{3\times3}$ performs geometrical rotations of a vector $\boldsymbol{x}$ in 3D which is an operation of three degrees of freedom that imposes the following constraints on R:

$$\begin{aligned} RR^{-1} &= I \\ det(R) &= 1\,. \end{aligned} \tag{2.26}$$

For avoiding unnecessary constraints or leveraging known data for the rotation, it may be convenient to represent the orientation of the pose or camera by an alternative representation of fewer degrees of freedom.

One such representation is by *Euler angles* which consist of three parameters $\{\psi, \theta, \phi\}$, which describe three successive rotations around each of the three coordinate axes most often with the convention of the Z-Y-X order. For a given set of Euler angles, R is given by

$$\begin{aligned} R &= R_z(\phi)R_y(\theta)R_x(\psi) \\ R_z(\phi) &= \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ R_y(\theta) &= \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \\ R_x(\psi) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\psi) & -sin(\psi) \\ 0 & sin(\psi) & cos(\psi) \end{bmatrix}. \end{aligned} \tag{2.27}$$

Another representation is *axis-angle*, which describes rotation about an axis defined by a unit vector $\hat{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$ with an angle θ.

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix} \tag{2.28}$$

When solving for the pose or camera matrix, the solution is carried out by making the substitution $R = R(\psi, \theta, \phi)$ or $R = R(u_x, u_y, u_z, \theta)$ for the two cases respectively in (2.14).

2.1.4 Pose optimization with plane constraint

When an object is lying on a flat surface, the scenery can be modeled as an object model being attached to a plane. When solving the PnP problem, this can be exploited by constraining the solution, for which the degrees of freedom are reduced from 6 to 3. Such a plane can be inferred from an object with known pose, and which is known to also be attached to the same plane, by the following procedure.

Consider the points $\{\boldsymbol{X}_k\}_{k=1}^K$ of the object model which are known to be attached to the plane. The global location $\{\boldsymbol{X}'_k\}_{k=1}^K$ of these points are given by using the known pose

$$\boldsymbol{X}'_k = R\boldsymbol{X}_k + \boldsymbol{t}\,. \tag{2.29}$$

The attached plane π is then acquired by fitting the coordinates $\{\boldsymbol{X}'_k\}_{k=1}^K$. The plane may be represented by a single point p_0 belonging to it together with either the plane normal n or two vectors $\{u, v\}$ spanning it. The plane is used to constrain the pose problem in the following way.

Consider another object with unknown pose but for which the model points $\{\tilde{\boldsymbol{X}}_l\}_{l=1}^L$ that are attached to the plane π are known in the object specific coordinate system. A plane $\tilde{\pi}$ is fitted to $\{\tilde{\boldsymbol{X}}_l\}_{l=1}^L$ in the object coordinate system and represented by $\tilde{p_0}$, $\tilde{n}$,$\{\tilde{u}, \tilde{v}\}$. Note that the distance $\tilde{d}$ between $\tilde{\pi}$ and the origin is $\tilde{d} = \tilde{p_0}^T\tilde{n}$ and is a known quantity. This plane is expected to align with the known plane π when the object is transformed by the correct pose since it is known that

$$R\tilde{\boldsymbol{X}}_l + \boldsymbol{t} \in \pi \;\forall\; \tilde{\boldsymbol{X}}_l \tag{2.30}$$

This imposes the following two constraints on the PnP problem:

$$\tilde{p}' = R\tilde{p} + \boldsymbol{t} \in \pi \;\forall \tilde{p} \;\in \tilde{\pi} \tag{2.31}$$

$$R\tilde{n} = n\,. \tag{2.32}$$

These constraints can be used to remove one DoF for $\boldsymbol{t}$ by writing it on the form $\boldsymbol{t} = \tilde{p}' + nd$, and inferring d which is the orthogonal distance to π according to

$$\boldsymbol{t} = p_0 + nd \implies \tag{2.33}$$

$$nd = \boldsymbol{t} - p_0 \implies \tag{2.34}$$

$$d = (\boldsymbol{t} - p_0)^T n = (R\tilde{p_0})^T n = \tag{2.35}$$

$$\tilde{p_0}^T R^T n = \tilde{p_0}\tilde{n} = \tilde{d}\,. \tag{2.36}$$

The constraint $R\tilde{n} = n$ can be exploited in the following way. Note that a particular solution to the equation is given by representing R with an axis-angle rotation $R_1(\tilde{n})$ with the parameters $\hat{u}_1 = \tilde{n} \times n$ and $\theta_1 = \arccos \tilde{n}^T \cdot n$. This solution is not unique however since any subsequent rotation $R_2(n)$ around the n axis, i.e. $\hat{u}_2 = n$, will still fulfill the equation. R is therefor represented as the two sequential axis-angle rotations

$$R(\tilde{n}) = R_2(R_1(\tilde{n})) = R_2(n)\,. \tag{2.37}$$

Since there is only a single unknown parameter θ in this function, the DoF are 1.

2.1.5 Inferring the image projection

Utilizing the pose transformation (2.17) for a known pose and a particular object model point, it is possible to predict the placement of the image projection for any image in the motion object. The expected image point in a camera is given by

$$\pi(P'_p \boldsymbol{X}) = \pi\left(P' \begin{bmatrix} R^{-1} & -R^{-1}\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_p & \boldsymbol{t}_p \\ 0 & 1 \end{bmatrix} \boldsymbol{X}\right) \tag{2.38}$$

This can for instance be used in conjunction with object-models to predict which pixels in an image will belong to the object. Although the simple procedure of projecting all object points into an image does not incorporate the possibility of objects being occluded, more intricate algorithms exist which accurately predicts object pixels. The output of such an algorithm could both be rendered images (in the case of colored meshes of objects), and instance segmentation data for each object.

2.1.6 Object model representation and keypoints

An object can be modeled by a *point cloud* which is a collection of N 3D coordinates $\{\boldsymbol{X}_i\}_{i=1}^{N}$ (usually labeled "vertices") defined in an object specific coordinate system [24]. The coordinate system is usually defined to have its origin in the geometrical centre of the object, and axes such that the z-axis points in a direction that one could naturally think of as the "up"-direction of the object (although this choice is arbitrary).

The diameter D of a point cloud is defined to be the maximum Euclidian distance between any two points in the point cloud [25].

In addition to the vertices, a model may also have a set of faces $\{F_i\}_{i=1}^{F}$ which define a piece-wise continuous surface of the object. Each face F_i is defined by a number of vertex indices which span a simple geometric surface shape (usually 3 vertices

spanning a triangle). Having model faces defined is important when rendering 3D models in an image, to fill the space between projected image points.

In addition to the point cloud, the model may also be accompanied in some scenarios by a set of K *keypoints* $\{\boldsymbol{X}_k\}_{k=1}^{K}$ also defined in the object specific coordinate system. Note that keypoints may also be defined as indices of certain points in the already existing point cloud. Properly sampled keypoints may serve to simplify pose estimation problems by reducing the number of 2D/3D point correspondences needed to solve for the pose [26]. Furthermore, properly sampled keypoints may also have semantic indication in the object model surrounding them, which can be exploited in learning based approaches [20] [27] [28]. Keypoint sampling is usually carried out either in an efficient automated fashion, such as farthest point sampling [20], or by manually selecting keypoints which appear useful, such as sharp edges or distinct textures [29].

2.2 Deep learning

Artificial Neural Networks (ANN) most commonly refer to input-output models inspired by the human brain [30] which act as universal function approximators [31]. The ANN architecture consists of a number of *neurons* which are sequentially connected, each of which accepts a multi-dimensional input and produces a one-dimensional output. See Figure 2.2 for a visual representation of a standard feed-forward neural network model. Deep learning refers to ANN's which are "deep", i.e. consisting of a large number of neurons and layers.

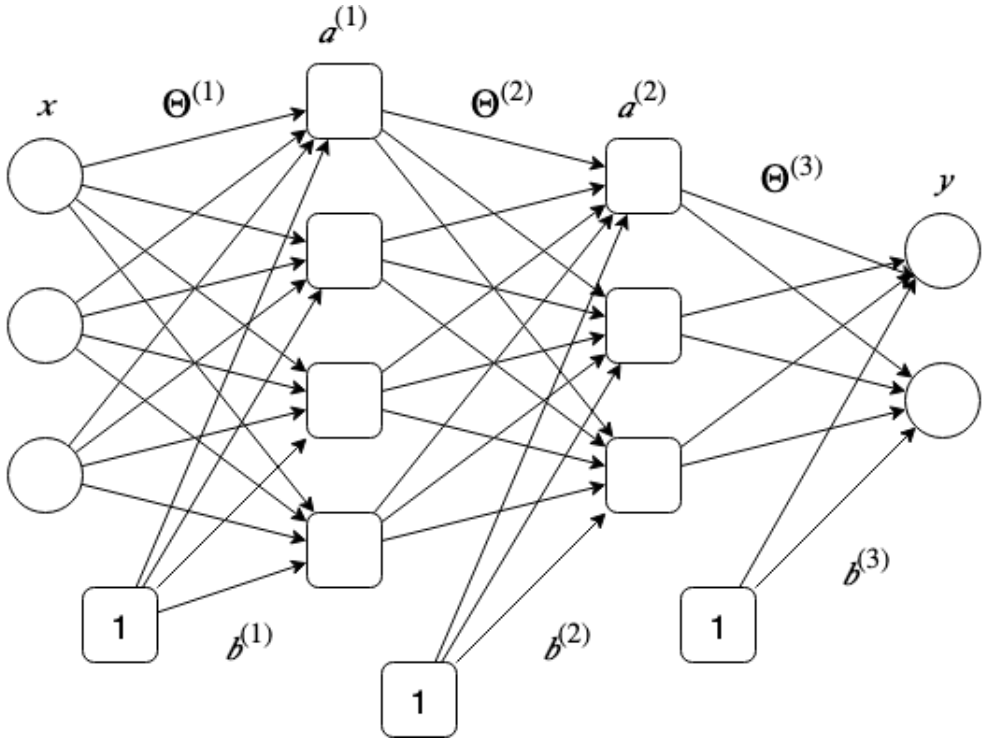


Figure 2.2: Schematics of a feed-forward artificial neural network.

The most common neural network model, the Feed Forward neural Network, consists of L sequential layers, where layer l contains N_l neurons. First, a weighted linear combination $\boldsymbol{z}^{(1)}$ is computed from the input $\boldsymbol{x} \in \mathbb{R}^d$ by multiplication with a weight matrix $\Theta^{(1)} \in \mathbb{R}^{d \times N_1}$ and addition of a bias vector $\boldsymbol{b}^{(1)} \in \mathbb{R}^{N_1}$ (2.39). The activation $\boldsymbol{a}^{(1)}$ of the first layer is then given by the activation function $f_a(\boldsymbol{z}^{(1)})$ (2.40). The activation function serves the important purpose of introducing non-linearity to the model. Without a non-linear activation function, the final output will only depend linearly on the input, limiting the complexity and applicability of the model [32]. Popular choices of activation functions include ReLU [33] and the hyperbolic tangent function. For each of the subsequent layers, the same two steps are taken, but with the activation of the previous layer as the input rather than $\boldsymbol{x}$ (2.41). The output of $\boldsymbol{y}$ is simply taken to be the activation of the final layer $l = L$.

$$\boldsymbol{z}^{(1)} = \Theta^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \tag{2.39}$$

$$\boldsymbol{a}^{(l)} = f_a(\boldsymbol{z}^{(l)}) \tag{2.40}$$

$$\boldsymbol{z}^{(l)} = \Theta^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{2.41}$$

Deep learning models are commonly used to model unknown systems $\boldsymbol{f}(\boldsymbol{x})$ for which data in the form of N input-output pairs $\left\{(\boldsymbol{x}, \boldsymbol{y}_t)^{(i)}\right\}_{i=1}^{N}$ are given. The subscript t denotes that this is the target output. By tweaking the parameters $\boldsymbol{\theta} = \left\{\left(\Theta^{(l)}, \boldsymbol{b}^{(l)}\right)\right\}_{l=1}^{L-1}$ of the neural network $\hat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{x})$, the complexity of a sufficiently

deep neural network allows for the model to match a vast set of input-output pars to a high degree. The parameters are tweaked by gradient-based methods such as Stochastic Gradient-Descent, or more sophisticated methods such as ADAM [34], minimizing an appropriate objective function $E(\boldsymbol{\theta}; \boldsymbol{y}_t)$ (loss function). To compute the gradient of the loss function w.r.t $\boldsymbol{\theta}$, the backpropagation algorithm is used. The details of backpropagation is omitted here, but it essentially boils down to applying the chain rule in several steps, assuming the loss function is differentiable [35].

The appropriate loss function varies from task to task, but is always chosen such that outputs produced by $\hat{\boldsymbol{f}}_{\boldsymbol{\theta}}$ which are similar to $\boldsymbol{y}_t$ yield lower loss values. For example, in the task of binary classification, the loss function is usually taken to be the Cross-Entropy loss (2.42). Note that the dependence of $\boldsymbol{\theta}$ is incorporated into the output $\boldsymbol{y}$.

$$E(\boldsymbol{\theta}; \boldsymbol{y}_t) = -\frac{1}{N}\sum_{i=1}^{N} y_t^{(i)} \cdot \log\left(y^{(i)}\right) + \left(1 - y_t^{(i)}\right) \cdot \log\left(1 - y^{(i)}\right) \tag{2.42}$$

Deep learning has been successfully deployed in numerous tasks within computer vision, such as image depth estimation [36] and object classification [37], where the deep learning models outperformed the traditional methods. Deep learning based methods have the ability to capture semantics in data to solve tasks for which humans have no difficulty solving (e.g. stating whether or not an image contains a dog), but for which it is very difficult to formulate a rule-based algorithm which solves it.

2.2.1 Convolutional neural networks

A Convolutional Neural Network (CNN) is a deep learning model which processes data through convolutions. CNN's are particularly useful within Computer Vision, where the grid-like structure and spatial coherence of images is exploited [32]. Specifically, this is accomplished by *weight sharing* which reduces the number of parameters of the network. Similar to the Feed Forward neural Network, a CNN successively performs a linear operation on the input $\boldsymbol{x}$ followed by an activation function and repeats these two operations a set number of times. The matrix multiplication $\Theta^{(l)}\boldsymbol{a}^{(l-1)}$ in equation 2.39 is replaced by a discrete convolution operation, see Figure 2.3 for a visualization of this operation.

In the case of images, the input is usually a tensor of dimension $(h_x \times w_x \times d)$ which is the height,width and number of channels of the image. The discrete convolution is a function associated with a tensor K denoted a *kernel*, which in this case is of dimension $(h_K \times w_K \times d)$. The convolution is performed by taking the inner product between the two tensors $\boldsymbol{x}$ and K in a sliding window fashion. The elements of the output tensor $\boldsymbol{z}$ are defined by

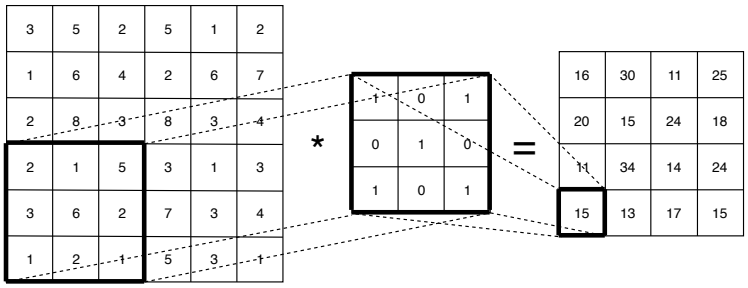


Figure 2.3: Depiction of discrete convolution between an image (6×6) and a filter (3×3).

$$\boldsymbol{z}(i,j) = \sum_{m,n,o} K(m,n,o) \cdot \boldsymbol{x}(m+i-1, n+j-1, 0) \tag{2.43}$$

$$m = 1, 2, ..., h_K \tag{2.44}$$

$$n = 1, 2, ..., w_K \tag{2.45}$$

$$o = 1, 2, ..., d \tag{2.46}$$

$\boldsymbol{z}$ is of dimension $((h_x - h_K + 1) \times (w_x - w_K + 1) \times 1)$. Usually the input $\boldsymbol{x}$ is *padded* in such a way that $\boldsymbol{z}$ gets the same height and width as $\boldsymbol{x}$. Padding means that the input is extended in the relevant dimensions by adding zeros in the beginning and end of it.

A typical layer in a CNN consists of n different kernels $\{K\}_{k=1}^{n}$, where n may differ between the layers. Each of the kernels produces an output $\boldsymbol{z}_k$ which are stacked together to produce the complete output tensor $\boldsymbol{Z}(i,j,k) = \boldsymbol{z}_k(i,j)$ for a layer. A bias vector $\boldsymbol{b}$ is added to $\boldsymbol{Z}$ and used as input to the activation function similarly as in the Feed Forward case.

CNN's have been proven efficient in object detection on pixel level, i.e. *segmentation problems* [38]. The segmentation problem consist of classifying pixels into a number of categories, of which one usually represents background or lack of an object.

2.3 Object detection and 6D pose estimation

The problem of object detection is the problem of identifying n objects where each object is labeled as one of m classes, and finding their locations in some space, given some input. The input contains no information regarding the inventory of objects. (As opposed to the related problem of object localization [39]) In the more researched area of 2D object detection, the location is typically represented by a bounding box in the image, defined by x and y coordinates of a corner, and side lengths h and w [40] [41]. In 6D pose estimation and object detection, the location is represented by the 6D pose relative some reference camera.

The ground truth data G is specified as a set of n objects O_i which are defined by one of m available class labels, and a pose P_i.

$$G = \left\{ O_i = \left(\text{class}_i \in \{C_j\}_{j=1}^{m} \, , \, P_i \in \mathbb{R}^{3\times 4} \right) \right\}_{i=1}^{n}$$

The goal of a 6D pose estimation and object detection algorithm is to, given some input data x, produce a detection set D which is specified as a set of n' objects O'_i, defined as for the ground truth data.

$$D = \left\{ O'_i = \left(\text{class}_i \in \{C_j\}_{j=1}^{m} \, , \, P_i \in \mathbb{R}^{3\times 4} \right) \right\}_{i=1}^{n'}$$

Usually one includes a "confidence" metric to the output objects O'_i in addition to the pose and class label. Confidence can be used as an intermediate step in filtering detections before deciding upon the final set D [42], and the confidence of the final detections are typically used while calculating one of the most popular evaluation metrics, Mean Average Precision [39].

2.3.1 Evaluation

An object detection algorithm is evaluated by comparing the detection set D to the ground truth set G.

A detected object O'_i in D is referred to as *matched* with an object O_i in G if they belong to the same class and are sufficiently close according to some metric as described in Section 2.3.1.2. Depending on the choice of metric, there may also be multiple detected objects $\{O'_i | i = 1, 2, ..., k\}$ matched with the same ground truth object or vice versa. Object detection algorithms are typically evaluated using classification metrics [43] [44] where the number of *true positives* (TP), *false positives* (FP) and *false negatives* (FN) are counted by the following definitions.

A FP is counted for each object in the detection set D that is not matched with any object in G. A TP is counted for each matched detection O'_i, in which case it is said to be *assigned* to the ground truth object it is matched with, except in the case of multiple detections $\{O'_i | i = 1, 2, ..., k\}$ being matched with the same ground truth object O_i. In this case only one of the detections O'_i will be counted as a TP, and furthermore assigned to O', and the rest will be counted as FP. For each object in G that was not matched, a FN is counted. In typical classification evaluation, there is also the concept of *true negatives*, but this is not applicable for the object detection problem, since there are an infinite amount of locations where there is no object in a scene.

One of several evaluation metrics for pose detection is the F1-score, which may be written on the form

$$F_1 = \frac{2TP}{2TP + FP + FN} = \frac{TP}{n + n'} \tag{2.47}$$

where n and n' are the number of objects in the sets G and D respectively.

2.3.1.1 Assigning the matches

There is an ambiguity when using the F_1-score for this problem which is a consequence of the fact that TP will depend on how the object assignment is performed. Since the choice of assignment is ambiguous, different approaches exist which specify how it is to be carried out. One approach is by assigning the detection matches in the most profitable way as to maximize TP [39]. This approach result in a variation of the *assignment problem* [45], for which multiple algorithms provide the solution [45] [46] [47]. One way of assigning the matched instances in the most profitable way is by the use of the *Hungarian Algorithm* [45] with a binary cost matrix; Either detection i is a match with ground truth j or it is not.

2.3.1.2 Matching between ground truth and detections

The concept of "closeness" can be defined in different ways depending on the problem. In 2D object detection, a detected object is taken to be sufficiently close to the ground truth if the *Intersection over Union* (IoU) between their bounding boxes is greater than a set threshold (typically 0.5 or 0.9) [48]. The metric is simply the fraction between the area of the intersection of the boxes, and the area of the union of the boxes. For our problem of detection of objects in 3D space, the IoU metric generalizes by using 3D bounding boxes which encloses the CAD-models of the objects. Alternatively, one can use other metrics such as the 5°-5cm metric [49] where a pose is correct if it is within 5° rotational and 5cm translational error. Another alternative is ADD which is the average distance of CAD-model points transformed with the estimated and ground truth poses [2], for which the threshold of detection typically is 10% of the object diameter.

2.4 Clustering algorithms

Clustering is the problem of finding structure in data and dividing the data into a number of groups according to some measure of similarity. The intention might be to categorize the data or acquire a more sparse representation of it. Different clustering algorithms may furthermore depend on carefully selected parameters to function properly.

2.4.1 K-Means clustering

K-Means is an algorithm for clustering data $\{\boldsymbol{x}_i | i = 1, 2, ..., n\}$ into a predefined K number of clusters $\{C_k | k = 1, 2, ..., K\}$. The idea is to find the means $\{\boldsymbol{\mu}_k | k = 1, 2, ..., K\}$ of the data clusters and group the data according to the closest mean point they belong to. Different measures of distance may be applied and in this section the euclidean distance $d = ||\boldsymbol{x} - \boldsymbol{\mu}||_2$ is used.

The K-Means algorithm consists of the following steps

Result: A set of K means
Initialize K means $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$;
while *not converged* **do**
  for $\boldsymbol{x}_i \in \{\boldsymbol{x}_i\}_{i=1}^{n}$ **do**
    Calculate k from $\min_k ||\boldsymbol{x}_i - \boldsymbol{\mu}_k||_2$
    Set $\boldsymbol{x}_i \in C_k$
  end
  if $\{C_k | k = 1, 2, ..., K\}$ *equal to last iteration* **then**
    converged
  end
  for $\boldsymbol{\mu}_k \in \{\boldsymbol{\mu}_k\}_{k=1}^{K}$ **do**
    $\boldsymbol{\mu}_k = \frac{1}{S_k} \sum_{\boldsymbol{x} \in C_k} \boldsymbol{x}_i$
    $S_k = \sum_{\boldsymbol{x} \in C_k(1)}$
  end
end
Different schemes exist for initialization of $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$, a common one being random selection in the set of data;

Algorithm 1: K-means algorithm

2.4.2 Mean-shift clustering

The Mean-shift algorithm is a method used to find modes of a density function from discretely sampled data of the distribution [50], and can easily be extended to be used for clustering tasks [51]. Mean-shift has been used for tasks in computer vision such as segmentation or object tracking [52].

The underlying idea of the method is to estimate the gradient of the density function $f(\boldsymbol{x})$ at point $\boldsymbol{x}$ to be proportional to the *mean-shift* $\boldsymbol{m}(\boldsymbol{x})$. By initializing a number of *centroids* in the feature space and iteratively taking steps in the estimated gradient direction, properly initialized centroids will converge to the various modes of the distribution.

The mean-shift may be defined as difference between $\boldsymbol{x}$ and the mean of the samples

weighted by a kernel $K(\boldsymbol{x}_i - \boldsymbol{x})$ in the neighborhood $N(\boldsymbol{x})$. (2.48)

$$\boldsymbol{m}(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_i \in N(\boldsymbol{x})} K(\boldsymbol{x}_i - \boldsymbol{x})\boldsymbol{x}_i}{\sum_{\boldsymbol{x}_i \in N(\boldsymbol{x})} K(\boldsymbol{x}_i - \boldsymbol{x})} - \boldsymbol{x} \tag{2.48}$$

The kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$ is often chosen to be a flat kernel with bandwidth h, in which case the neighborhood $N(\boldsymbol{x})$ is automatically inferred, and the mean of points within the neighborhood is unweighted.

$$K(\boldsymbol{x} - \boldsymbol{x'}) = \begin{cases} 1 & \text{if } \|\boldsymbol{x} - \boldsymbol{x'}\| < h \\ 0 & \text{otherwise} \end{cases} \tag{2.49}$$

Intuitively, the idea behind using the mean-shift on data sampled from a density function as an estimate of the gradient is motivated by the expectation of observing more samples in a neighbourhood around $\boldsymbol{x}$ where the density is greater. Furthermore, it has been mathematically proven that the mean-shift at $\boldsymbol{x}$ is proportional to a kernel gradient density estimate under reasonable assumptions. The Mean-shift algorithm for mode detection is described below:

Result: A set of m mode coordinates
Initialize n centroids $\{\boldsymbol{x}_i\}_{i=1}^n$;
for $\boldsymbol{x}_i \in \{\boldsymbol{x}_i\}_{i=1}^n$ **do**
  while *not converged* **do**
    Calculate $\boldsymbol{m} = \boldsymbol{m}(\boldsymbol{x}_i)$ according to eq. (2.48);
    Set $\boldsymbol{x}_i \rightarrow \boldsymbol{x}_i + \boldsymbol{m}$;
    if $\|\boldsymbol{m}\| < \epsilon$ **then**
      converged
    end
  end
end
Filter out duplicates among n final centroid locations to obtain $m \leq n$ mode coordinates;

Algorithm 2: Mean-shift algorithm

A strength of the Mean-shift algorithm is the lack of parameters. With the most common choice of kernel functions, only one parameter h, the bandwidth of the kernel needs to be decided upon. This parameter has a physical interpretation for many applications [50], making the decision easier.

3

Project disposition & dataset acquisition

Since the purpose of the project was partially object detection in a supermarket setting, and partially development of a data Annotation Tool, a natural inclusion into the project was the creation of an annotated dataset of labeled object poses in a supermarket setting. That way, the development of the Annotation Tool could benefit from continuous testing/iteration, our detection algorithm could be trained and tested on a new dataset, and we would produce a dataset relevant to the aforementioned research project. A flowchart of the project disposition is shown in Figure 3.1.

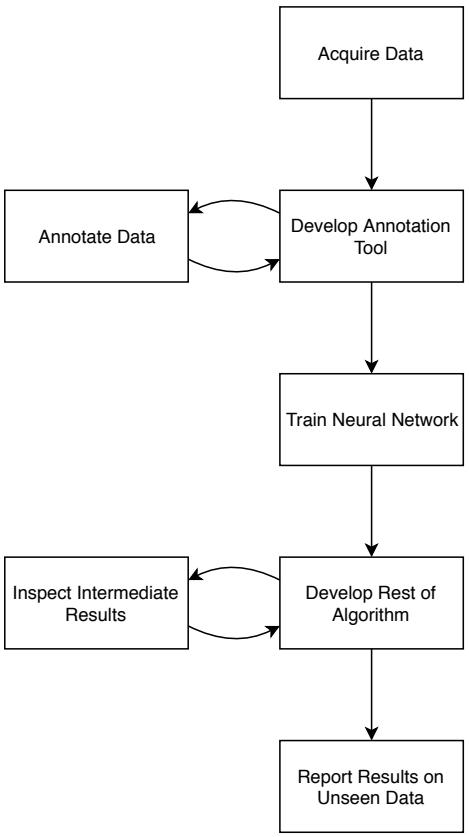


Figure 3.1: Flowchart diagram of project tasks.

3.1 Project planning and execution

The project started out by an extensive literature study which would shape the structure of the project. The goal of the study was partly to decide upon a state-of-the-art, deep-learning based method for single-view 6D pose estimation and object detection which we could extend to a multi-view framework. The method had to satisfy our constraint on legal inputs to our final algorithm (RGB images, object models and motion), and we also needed to be able generate the corresponding labels for the trainable part of the method.

Among the existing literature, one method, PVNet by S. Peng et al. [20], stood out for several reasons. Particularly due to their state-of-the-art performance on a well known dataset. We decided to build upon their work since a) they had source code published, b) their deep learning component produced interesting output which could enable flexibility, c) they claimed their method to be robust to occlusions and truncations, d) they proposed a method for extending their algorithm to enable detection of multiple instances, e) their performance on the popular occluded-LINEMOD Dataset [2] [53] was among the best reported at the time, and f) their model required labeled data which we would be able to produce with little effort from the labels generated using our tool.

The development of the Annotation Tool complemented the labeling of our dataset well, as we (expectedly) encountered bugs and flaws of the tool in practical use. Furthermore, we were able identify and to implement additional features which would make the software more efficient.

The finalized dataset was partitioned into a set of training scenes, and a set of validation scenes. The former set was used during development of the Detection Algorithm, manual parameter tweaking and deep learning model fitting. The latter set was used to assess the performance of the algorithm on new, unseen data, the values of which are reported.

3.2 Dataset creation

To acquire our datasets, we placed various objects in different environments of varying lightning, and recorded the scenes with a video camera.

The dataset included objects one could typically find in a supermarket setting, and also exhibited the same type of occlusions one could expect. The final list of objects used in our datasets included: Soap (green), Soap (blue), Paper cup, Box of Lentins, Box of Macaroni, Deciliter Measure, Can of Coconut Milk. The types of occlusions we took into account during creation included:

- Objects (partially) out of frame.

- Objects occluded by other inventory objects. This is common in supermarket shelves where items are typically lined up.
- Objects occluded by the environment. This could typically be the shelves themselves from some viewing angles.

These types of occlusions need to be handled differently in some scenarios, e.g. when producing segmentations by rendering objects. Note that our dataset lacks images with environmental occlusions due to the lack of a proper method to produce accurate segmentation labels taking that into account. That said, the Detection Algorithm may be able to produce decent results on test data in which environmental occlusions are present, if it is capable of dealing with the other types of occlusions.

Furthermore, each scene in the dataset was labeled with a "difficulty" with respect to occlusions to enable discussion of how occlusion affected the performance of our algorithm. Each scene was labeled as "Easy", "Medium" or "Hard" depending on the fraction of objects for which the video lacked viewpoints where they were clearly visible. For example, scenes most resembling an actual store shelf would be labeled as difficult, since objects are typically stacked tightly in a row, limiting the visibility of most items greatly.

The motion was calculated using the structure-from-motion framework COLMAP. The COLMAP pipeline required only the RGB images to calculate this.

The object models were obtained either by creating simple geometric shapes programmatically (for cuboids and cylinders), or by scanning them with HandyScan3D [54] and processing the data in VXmodel [55].

3.2.1 Resulting dataset

In Tables 3.1 and 3.2, statistics for the resulting datasets are presented. In total, data for 22 scenes were generated, 19 of which are used as the training set, and 3 of which as the validation set. The difficulty distribution of the training set is uniform, whereas the validation set consists of one medium, and two hard.

Unfortunately, many of the recorded scenes had to be discarded due to COLMAP either failing to generate motion objects which remotely represented the camera motion, or generating motion objects which simply were not accurate enough. Furthermore, among the scenes included in the datasets, many of the images were inexplicably ignored by COLMAP, resulting in fewer images on which to train the deep learning model. We approximate that 75% of the actually recorded frames were lost.

A sample of the scenes can be seen in Figure 3.2. For an overview of all the scenes, Figure B.1 in Appendix B presents a single frame for every scene which aims to give a good overview.

Table 3.1: Statistics of training dataset. The final 7 columns are the number of instances of each class as indexed in Table 3.3.

Idx	#Images	#Objects	Difficulty	#1	#2	#3	#4	#5	#6	#7
1	555	13	Easy	2	2	2	3	0	2	2
2	345	35	Hard	5	6	7	3	1	5	8
3	267	52	Hard	6	10	9	8	6	5	8
4	227	20	Easy	4	3	3	3	4	3	0
5	201	13	Easy	0	3	3	0	4	3	0
6	611	52	Hard	6	10	9	8	6	5	8
7	364	23	Medium	4	4	3	4	0	5	3
8	536	11	Easy	2	2	2	2	1	0	2
9	400	29	Hard	4	4	3	7	4	4	3
10	463	30	Hard	4	6	0	6	3	5	6
11	422	30	Hard	4	6	0	6	3	5	6
12	346	16	Easy	4	4	0	0	3	3	2
13	569	17	Medium	0	2	0	4	4	4	3
14	387	12	Medium	0	2	0	2	3	2	3
15	529	19	Medium	4	4	2	2	1	3	3
16	637	21	Medium	3	3	2	4	5	2	2
17	528	52	Medium	6	10	9	8	6	5	8
18	656	13	Easy	0	2	0	2	4	2	3
19	165	36	Hard	6	5	7	4	1	5	8
-	**8208**	**494**	-	**64**	**88**	**61**	**76**	**59**	**68**	**78**

Figure 3.2: Two of the scenes in the dataset.

Table 3.2: Statistics of validation dataset. The final 7 columns are the number of instances of each class as indexed in Table 3.3.

Idx	#Images	#Objects	Difficulty	#1	#2	#3	#4	#5	#6	#7
1	260	29	Medium	4	5	7	5	4	0	4
2	385	26	Hard	3	4	4	4	4	4	3
3	981	30	Hard	4	6	0	6	3	5	6
-	**1626**	**85**	-	**11**	**15**	**11**	**15**	**11**	**9**	**13**

Table 3.3: Table of classes used in our dataset, and their corresponding numeric indices for future reference.

Class Idx	Class Name
1	Deciliter Measure
2	Coconut Can
3	Paper Cup
4	Box of Macaroni
5	Soap Bottle (Green)
6	Soap Bottle (Blue)
7	Box of Lentils

4

Annotation Tool

This chapter covers the development of the Annotation Tool. It starts with an introduction and motivation of the core idea behind the tool, followed by a specification of user requirements and features. In the method section, the programming framework used to create the tool is presented, and the underlying mechanisms utilized are explained and motivated. Additionally, experiments which aim to evaluate the efficiency and quality of the core method are formulated. In the section labeled "Resulting Software", the features of the final product are presented along with discussion regarding usability of the tool. Thereafter, the results of the experiments are presented. Finally, the chapter closes with a section of discussion of the experiment results, and also regarding the quality of the method as a whole, with suggestions for future improvements and features.

4.1 Core idea and specification

Consider an RGB image sequence $\{I_t\}_{t=1}^{T}$ of a scene containing a number of *objects of interest* (OoI). The image sequence is accompanied by its *relative motion* $\{P_t\}_{t=1}^{T}$ as defined in Section 2.1.2.1. Assume the objects of interest belong to one of N object classes (e.g. Box of macaroni, Box of lentils, etc.). The goal is to obtain, for each image I_t in the sequence, the poses relative the image camera of all objects of interest specified with the class of the object as described in Section 2.3. Note that this includes poses of objects relative a camera for images in which the object may not be visible.

The core idea of the Annotation Tool is for the user to manually identify and label the poses of each object in the scene once, by identifying keypoints of the objects in a small subset of the images. The global pose of an object, which can be transformed to poses relative any camera, is calculated using reprojection error minimization across the different viewpoints. The user continuously assesses the quality of the pose annotations by observing the projections of the models onto the images from multiple viewpoints.

The utility of the tool lies in its ability to produce thousands of annotated images

from the manual effort of partial annotation in only a few, while allowing the user to seamlessly get an overview for the quality of fit of all labeled data. The tool makes no explicit assumptions regarding the environments in which the images have been captured, making it versatile for labeling new datasets for any new scenario.

The data provided by the user is the following:

- An RGB image sequence of the scene in question
- Pre-calculated camera motion
- The inner parameter matrix K of the camera which captured all images of the sequence
- Point cloud CAD models of the inventory objects, complete with vertices and faces
- (Optional) Keypoints associated with the CAD models

4.2 Method

4.2.1 Software development framework

The Annotation Tool was built using the MATLAB App Designer. This choice of platform was motivated by a number of reasons. Firstly, we had great familiarity with the MATLAB programming language, and the it includes efficient tool for matrix algebra and optimization, which are important concepts within computer vision. Secondly, MATLAB supports object-oriented programming, which provides flexibility when developing a user friendly tool. Finally, the App Designer allowed for easy user interface generation through its click-and-drag capabilities, which was especially useful considering our lack of experience with front-end development.

4.2.2 Underlying mechanisms

4.2.2.1 Defining the global coordinate system

In order to simplify things, it is useful to define a global coordinate system. Therefore, before any calculations are made in the tool, the motion object is transformed to a coordinate system where the first camera P_1 is the identity camera $[I, 0]$. We denote the global pose of an object as P_{pose}, and it is expressed in this coordinate system.

The provided motion object is not necessarily defined in a coordinate system which shares the scale of the object specific coordinate systems. (Recall that scale is ambiguous from Section 2.1.2.1) As a result, the pose P_{pose} of an object when transformed to a viewpoint t will be improperly scaled as explained in Section 2.1.3.1.

Therefore, we also introduced a multiplicative *scaling factor* s which scales the global coordinate system such that it is expressed in the same units as the objects' coordinate systems. The scaling factor can either be set manually by the user, or it can be specified to be a free parameter in the optimization procedure.

When navigating the different viewpoints, it is important that the user is able to visually assess the quality of the pose in the current image. This is enabled by simply transforming the global pose to a pose relative the current camera according to eq. (2.17), and then projecting the model point cloud onto the image.

4.2.2.2 Pose optimization

For a single object instance, given a collection of 2D coordinates $\{\boldsymbol{x}_{t,k}\}$ where t is the viewpoint index, and k denotes the keypoint type, we optimize for a pose P_{pose} such that the multi-view reprojection error E_{multi} (2.14) of the keypoints are minimized in all image planes. The optimization is carried out every time the user updates the set of 2D keypoint locations in the step described in Section 4.3.2.

In the case that the scaling factor s is a free parameter, the translation component t_t of $P_t = [R_t, t_t]$ is replaced by st_t. Note that this implies that scaling factor optimization is only carried out with respect to one object instance at a time.

The optimization procedure, like most optimizers, requires an initial solution P_{pose}^{init}. This was obtained by using a 3-point minimal solver on three randomly selected point correspondences. The solver results in two pose candidates, from which the most appropriate one must be manually assessed. The 3-point minimal solver was written by Olof Enqvist of the Computer Vision Group at Chalmers.

Once an initial solution has been obtained, the optimization is carried out using the function `fmincon` which is available in MATLAB's Optimization Toolbox. For a new pose solution $P_{pose}^{*} = [R^*, t^*]$ to be valid, R^* must be a rotational matrix. To avoid implementing the rotational constraint (2.26) during optimization, we utilized an Euler angle representation of R (2.28) and optimized w.r.t the Euler angles.

4.2.2.3 Pose optimization with plane constraint

In many cases, multiple objects lie on the same flat environment surface. Some of our objects have parts which are flat and may rest with that part on the underlying environment. To aid in the annotation, we model the underlying surface as a plane and make use of the plane constraint defined in Section 2.1.4.

As explained in 2.1.4, we need a set of points $\{\boldsymbol{X}_k\}_{k=1}^K$ on the object model which are known to be attached to the plane. To acquire such a set we associate with each object a number of *lying positions* (such as a macaroni box standing upright) where a particular set of points is defined. In the case of macaroni in upright position, the bottom four corners of macaroni are taken to be such a set. We choose the set of points associated with the lying positions so that the planes they define always are orthogonal to one of the coordinate axes. This simplifies our problem since the normal vector to the plane in the object coordinate system will always be equal to one of the coordinate axis $\tilde{n} \in \{\hat{x}, \hat{y}, \hat{z}\}$. The user can then specify the lying position by specifying which coordinate axes is equal to the plane normal, e.g. $\hat{z}$ when the macaroni is standing upright. A plane in the global coordinate system is then acquired by fitting the set of points, as explained in Section 2.1.4, and is represented by p_0 and n. The user may also specify multiple such known objects, in which case the set of keypoints defining the plane is the union of the objects individual sets.

The plane can then be used to first detect the lying position. The pose problem is initially solved without the aid of the plane by the normal solver, which will yield an initial solution $P_i = [R_i \; t_i]$. One of the coordinate axes will be similar to n after transformation, i.e. $(R_i\tilde{n})^T n \approx 1$.

Since the transformed coordinate axes are equal to the rows of the transformation $R = RI = R(\hat{x}, \hat{y}, \hat{z})$, we can find the relevant coordinate axis by finding which row of R produces the larges inner product with n

$$\max_i R_i^T n \tag{4.1}$$

where R_i^T denotes row i of R. i identifies the index for the correct coordinate axis. Since $\tilde{n}$ is now known, we can use equation (2.37) and solve the problem for the single parameter θ in an axis-angle representation.

4.2.3 Experiment 1: Assessing quality of labels

In order to assure that the pose labels generated using the tool are reasonable for each of the images in $\{I\}_{t=1}^T$, we needed to efficiently measure their quality-of-fit across all data. Since no ground truth labels were available prior to the annotation, we opted for a visual inspection. This was done by rendering transparent segmentations of the CAD models onto each frame in every video, and checking that they overlapped well with their respective objects in the images. For this report, we include a random sample of the frames across all videos. Although the user should be able to assess the quality of their labels during annotation using the tool, some systematic errors in the method may not always be apparent while using the tool, or some implementation errors during the exportation of the labels may occur. A comprehensive visual inspection of the scene is necessary to confirm that the ground truth labels produced are reasonable.

4.2.4 Experiment 2: Assessing the efficiency of annotation

Since the tool revolves around the user manually identifying keypoint locations in images, the number of identifications per object instance measures in a sense the manual effort of generating the data with our tool. To provide ground for discussion of how our tool facilitates data labeling, we retroactively counted this quantity from the labeling of our dataset. The histogram of the number of identifications per object, and the number of viewpoints utilized are reported. Furthermore, this experiment was formulated after the annotation had been performed. Therefore, we were not influenced to deliberately minimize the visible effort, and the data is a better representation of a typical use case.

4.3 Resulting software

In this section, the final Annotation Tool is presented. The usage of each feature of the tool is explained along with a discussion pertaining to strengths and weaknesses from a user perspective. Screenshots of the various user interfaces the tool consists of are shown in Figures 4.1, 4.2, 4.3 and 4.4 where Figure 4.1 shows the main view. Note that some of the screenshots have been cropped to better fit the report.

4.3.1 Main view

In the main view, the user may navigate between the images $\{I_t\}_{t=1}^{T}$ and select the active object instance among the available. The active instance is the instance for which all operations are performed. The current image is the image in which the user may identify keypoints of the selected object by pressing "Click Points". The list of available object instances is initially empty. The user adds an instance of every object they identify in the images. Each object instance has an associated object class, with an associated CAD model and list of keypoints.

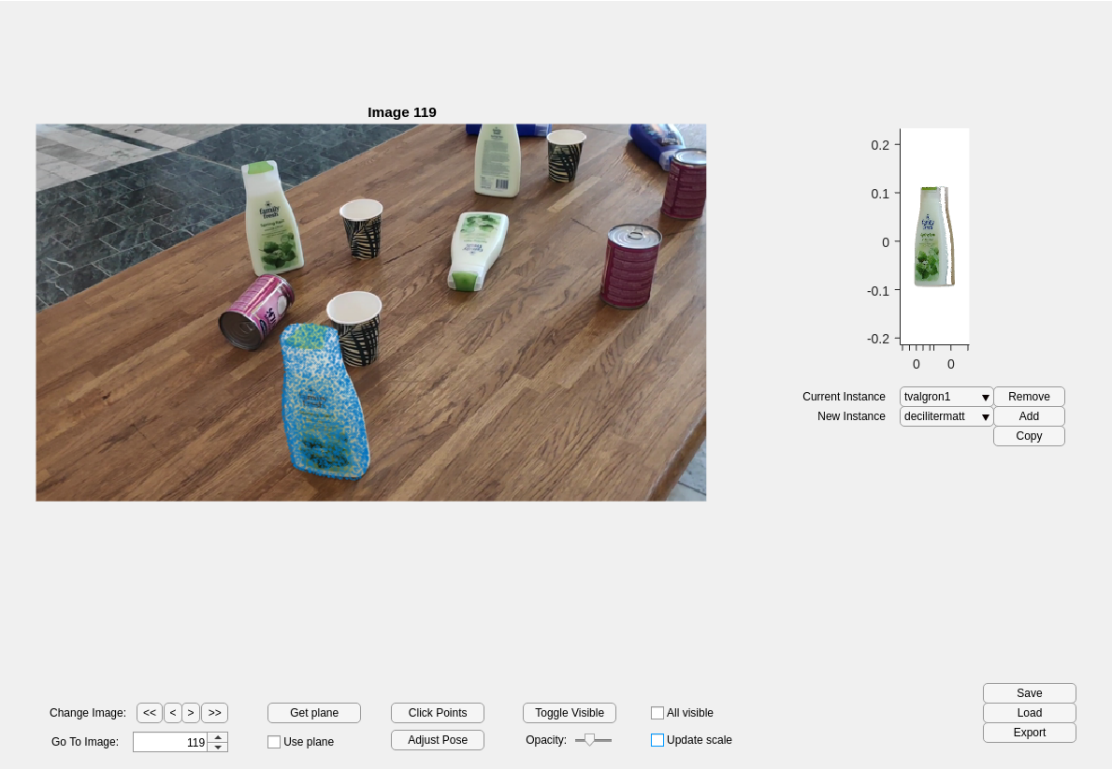


Figure 4.1: Main view after loading data of a scene. In this example, the projected point cloud of a green soap bottle is shown which was generated with a pose fitted by the user.

4.3.2 Point identification

The point identification window is opened when the user presses "Click Points" in the main view. In this window, the user is presented with a view of the CAD model of the selected OoI along with markers for its current set of keypoints. The user is also presented with the image of the selected viewpoint.

In this window, the user may select one of the defined keypoints as active by either selecting it from the list of defined keypoints, or by marking it graphically in the model view. When a keypoint is marked as active, the user may mark a pixel coordinate in the viewpoint image by simply clicking it. This creates a 2D keypoint coordinate $\boldsymbol{x}_{t,k}$ and adds it to the set of coordinates associated with the selected object instance.

The user is also able to add new object keypoints by marking them graphically on the CAD model. This is useful in case no keypoints have been defined a priori, or if the user discovers a new particularly helpful set of points. The new keypoints can also be exported for future use.

The user is able to use MATLAB's integrated tools for plot view manipulation, such

as rotating the CAD model or zooming and panning in the camera image, allowing for clearer views and more precise identification.

Upon confirming the identified set of 2D keypoint locations, the user is returned to the main view, and the pose is automatically optimized w.r.t all current keypoint correspondences associated with the model. If the model had no correspondences associated with it prior to this, two candidates for the initial solution for optimization are presented to the user. The user selects the best fitting of the two candidates in the view shown in Figure 4.3.

Figure 4.2: View where user may identify 2D keypoint locations $\{\boldsymbol{x}_{t,k}\}$ for arbitrary keypoint types k in a viewpoint t. Here, `keypoint5` (lower left corner) of the green soap bottle is the active keypoint, as indicated by the green color of the markers. Other keypoints marked with red markers have a 2D correspondence identified, and keypoints marked with black markers lack a 2D correspondence.

Figure 4.3: Prompt view to manually select one of two initial pose candidates. The user simply selects the pose which appears to be the better fit as the initial solution for optimization, which clearly is the rightmost solution in this case.

4.3.3 Manual pose manipulation

Once a pose has been obtained from the optimization procedure(s), the user should be able to manually fine adjust the pose to eliminate errors potentially introduced during identification. Since the projection of a point cloud onto one image plane can give a misleading idea of assessing the quality of the pose, it is important to be able visualize the projection in multiple viewpoints. Therefore, 4 viewpoints, evenly spaced in the image sequence, are presented in the same view to allow for better feedback during fine adjustment. The viewpoints of all 4 views can be navigated through.

The fine adjustment is carried out by selecting one of three coordinate system axes and either translating in the positive/negative direction, or rotating in the clockwise/counter-clockwise direction. The step size for either type of transformation can be adjusted using a slider. Transformation steps are taken using keyboard combinations for maximum efficiency.

Figure 4.4 displays the fine adjustment view. In the example scenario shown in the figure, one may think from the left uppermost reprojection onto the image that the current pose is a good fit. However, upon inspection of the projection onto other viewpoints, one can clearly see that this is not the case. This example demonstrates the usefulness of the multi-view display when manipulating the pose label. Without the ability to continuously assess the quality-of-fit of the projection in multiple viewpoints, one may focus too much on seemingly perfecting the projection in one view without realizing how much worse the projection fits in other views.

Figure 4.4: Fine adjustment view. The user is presented with 4 viewpoints focused on the selected object. The user selects one of three object axes (selected axis is green, other axes are red) which they can either rotate the pose about, or translate along with. The sliders labeled "Rotation" and "Translation" adjusts the step size of rotations and translations respectively.

4.3.4 Plane identification

As mentioned in Section 2.1.6 it is not uncommon for objects point clouds to be defined in a coordinate system where the axes are oriented along certain key directions pertaining to the object geometry. An example would be that the z-axis is defined to point in the direction one naturally would assume to be up if the object rested on a flat surface (such as the top of a table). E.g., the CAD models of the objects available in Hinterstoissers dataset from the SIXD challenge repertoire are expressed in coordinate systems abiding this rule of thumb.

Under this assumption, the tool allows users to select a number of the object instances for which poses have been fit. For each instance, the axis orthogonal to the plane and the direction of the axis pointing towards the plane are selected. The plane is then fitted to the instances, and the user may choose to toggle on or off the plane constraint described in Section 4.2.2.3 for new object instances.

After implementing the plane constraint, we found that the number of cases where

the poses required fine adjustment after the initial identification of keypoints was reduced significantly. This naturally reduces the workload of annotation in cases when there are numerous objects in natural resting positions on a flat surface, which was particularly common for our dataset emulating supermarket shelves.

In the final product, there is no visualization of the plane in the main view. This makes it difficult to assess the quality of the plane estimate. A simple additional implementation which projects a bounded rectangle or circle in the main view, with the ability to scale the bounds and translate the center, would have made for a better user experience. Furthermore, it could also be desirable to be able to define a plane without first needing to find the poses of some objects. Instead, a plane could be defined by allowing for the user to e.g. identify a set of points on the flat surface and locate their 2D coordinates in two or more viewpoints from which a plane can be fitted much like the poses.

4.4 Experiment results

The videos of all of the 22 scenes with object segmentation renders were inspected. From the resulting segmentations, the poses were deemed to be good labels. Randomly sampled frames of the produced videos are included in Appendix C for the reader to personally verify, see Figure C.1.

The recorded numbers of identifications during the annotation of our dataset is presented in Figure 4.5. The recorded numbers of viewpoints utilized is presented in Figure 4.6.

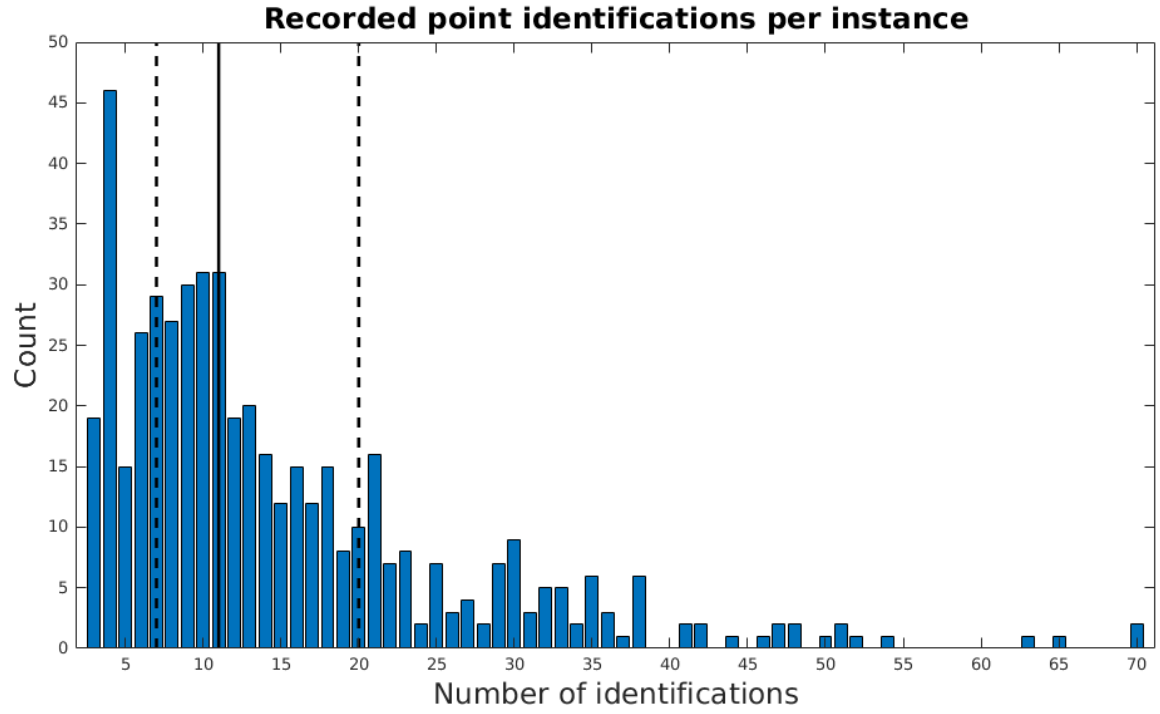


Figure 4.5: Histogram of the number of 2D points identified for fitting the pose of an object instance. The thick black bar denotes the median, and the left and right dashed bars denote the 0.25- and 0.75-quantiles respectively.

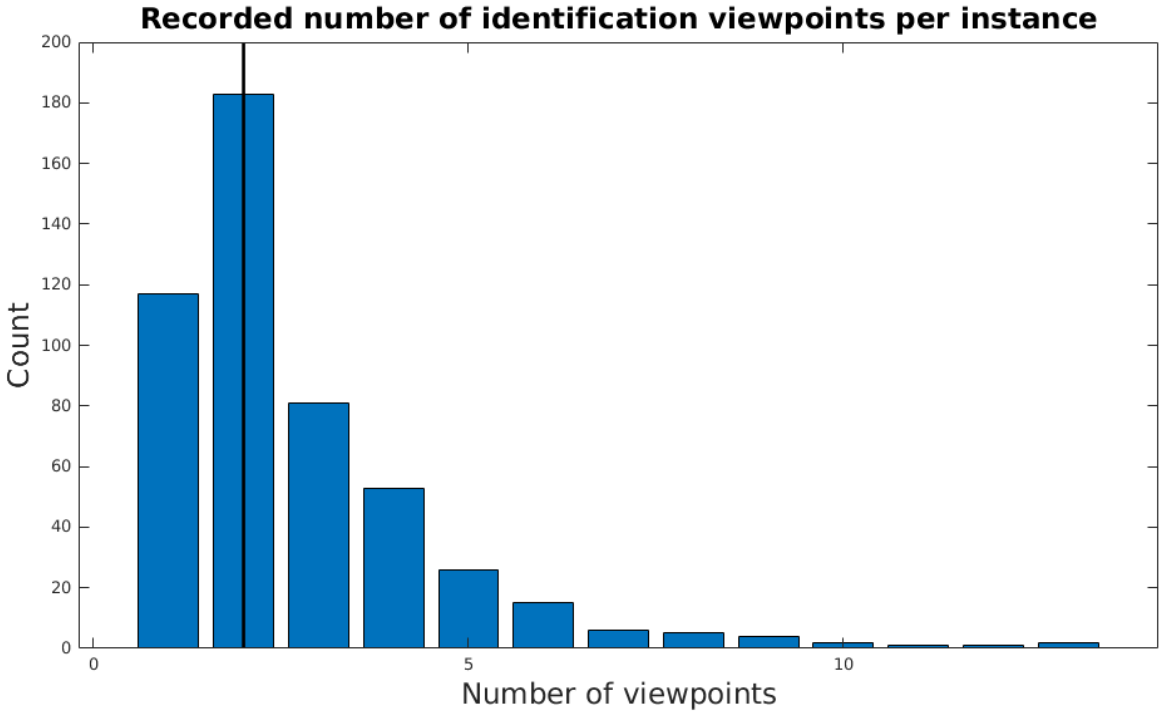


Figure 4.6: Histogram of the number of viewpoints utilized for fitting the pose of an object instance. The thick black bar denotes the median.

4.5 Discussion

The aim of this section is to summarize the the main features and discuss how they facilitate the annotation process. Also, the results of the experiments are discussed. Furthermore, we discuss flaws with the produces tool and conclude this section with proposals for future improvement.

4.5.1 Summary of key features

During annotation, an object might be heavily occluded from all viewpoints, with none or few of the predefined keypoints being visible in any image. The resulting tool provides a method for the user to define new object keypoints live during annotation by selecting the keypoint directly on the CAD-model, which is particularly useful when objects are textured.

The tool provides continuous assessment of the pose by easy navigation between images in conjunction with the projected point cloud of the pose. This is important since the quality of the current pose estimate might be hard to visualize from a single viewpoint. An overview of the annotation state is also provided by the feature for projecting all point clouds of annotated objects.

A pose can be manually refined by continuous translation/rotation with visualization in multiple views. This is particularly useful when the pose estimate is poor due to noisy clicked point or outliers, where it may hard to improve it by further clicking.

Functionality is provided for exploiting environments where objects are lying on a flat surface. As the pose problem is then constrained, the DoF are reduced and fewer click are needed.

The tool allows unscaled motion objects as input, since the scale can be determined as part of the optimization procedure. This may be useful when motion object are given directly from a software like COLMAP, which was the case for our own annotated dataset. Since we consider our dataset to be of sufficient quality, we draw the conclusion that this way of determining scale is feasible.

4.5.2 Workload analysis

We note that in the majority of cases, the number of used viewpoints is in line with our expectations. When a pose is calculated from a single viewpoint the solution suffers from noise that is particularly scattered along the direction between the camera and the object, i.e. along the translation vector t of the local pose. Identifying further points from a viewpoint which is orthogonal to the first one is expected improve the pose estimate significantly since it reduces the noise in the mentioned

t direction. A third viewing direction orthogonal to the first two is not expected to have the same effect, although it usually still means an improvement, since there is no longer a particular spatial direction of high noise. Given that the user has identified points carefully, two viewpoints is expected to be sufficient in many cases.

In contrast to above argument, we do in fact observe a significant number of single-view identifications. A possible explanation for this is the use of the plane feature, since the solution in this case is restricted to be on this plane. Any identifications made in a viewpoint from above the object will for this reason not suffer the increased noise since the viewpoint is orthogonal to the restrictive plane.

We further observe cases where the number of viewpoints are between five and fourteen, which is a larger number than desired. One reason for this is the case where an object is heavily occluded in all images, and only a small part of the object with only a few keypoints can be identified, in which case multiple viewpoints are needed. Note that the live adding of keypoints came in particularly handy in this case. Related to this is the lack of good texture for some objects. For instance, the paper cup texture is of particularly bad quality, with the consequence that only a few keypoints on the model could be defined. More effort placed on improving the texture for some of the inventory objects would certainly improve annotation efficiency in these cases.

As observed in Figure 4.5, a majority of the identifications (the 0.75-quantile) are below 20. We deem this to be sufficiently efficient for annotation since it approximately translates to 10 identifications in two viewpoints or 7 identifications in 3, although we consider this result to be improvable. The previously mentioned lack of texture quality could potentially have an impact since keypoints that exploited the texture might suffer from bad positioning on the object.

4.5.3 Visual assessment of pose label quality

A flaw with the method used to assess the quality of the final labels used is the inability to conclude to what degree the orientation of geometrically rotation symmetric objects is correct. For example, the paper cup in our dataset, whose "forward" orientation is encoded in the texture of the cup, has the same projection for all rotations about the z-axis. Assuming one wishes to find the 6DoF poses for such objects, allowing the texture to signify principal directions, the methods we used both in the tool, and in the assessment of our produced labels, is insufficient. One could potentially solve this problem by rendering a transparent mask with the texture of the object.

Despite the inability to correctly assess the rotation, should we expect the labels of our dataset to to be flawed? Provided that the correct 2D keypoint locations were identified, the fitted pose should be correct. However, due to the method we used to texturize the models, we are not confident that the textures precisely match

the textures of the actual object. This in conjunction with manual fine adjustment, which was likely carried out for many of the objects, introduces reasons to believe that the final pose labels for the rotationally symmetrical objects in our dataset (Coconut Can and Paper Cup) are flawed.

In the case of labeling objects which are rotationally symmetric, both geometrically and w.r.t texture, the core method of our tool is flawed. When identifying keypoints from different viewpoints, consider a keypoint along the surface of the object whose 2D location was identified as $\boldsymbol{x}_{k,t}$ in viewpoint t. It would be very difficult, if not impossible, for a user to find the correct 2D location $\boldsymbol{x}_{k,t'}$ of the same keypoint in a different view t'. As a result, an appropriate pose may not be found at all when optimizing with respect to points form multiple viewpoints.

With this said, during evaluation of pose estimation algorithms, one may use evaluation metrics which take into account rotational symmetries, and do not punish the algorithm for producing estimates whose orientation does not match the ground truth in this regard. If one decides to evaluate an algorithm on our dataset with such a metric, and decides not to use the model textures to define principal directions, the labels can still be used for evaluation.

4.5.4 Flaws of the tool

Since the tool is based on the transformation described in Section 2.1.3, it heavily depends on the usage of a motion object. Acquiring a motion object of sufficient quality may not be feasible for any given set of images. For instance, we used COLMAP for acquiring the motion, which was unable to calculate a sufficiently accurate motion object in all image sequences. A user may therefore be restricted in the data collection in some cases.

A restriction when exploiting the plane optimization is that only specific predefined parts of the object model may be attached to the plane. Furthermore, the plane defined by such parts are constrained to be orthogonal to a coordinate axis. More general object shapes which are not adapted to these requirements are not handled in the current implementation.

MATLAB App Designer proved to be slower than desired during usage. This is particularly noted during loading of new screens, during addition of new instances and during changes of viewpoints. Although convenient during development, the fact that MATLAB App Designer is not an A-grade software development program impairs the usability of the Annotation tool.

Finally, a problem which we have identified stems from the simplified camera model used for pose calculations. As reviewed in Section 2.1.1, the pinhole camera model does not take into account some distortions introduced when capturing images with a real camera. Particularly, the pinhole camera tends to be a less accurate mapping from the 3D point space to the image plane the further away from the viewing ray

the 3D points are. Therefore, 2D points indicated by a user close to the edges of image suffer from a directional bias, which may have a significant impact on the accuracy of the fitted pose. This effect was noticed when annotating our data. This is especially troublesome since a user may intuitively assume that their keypoint indication should be more accurate if the image was taken close to the object, since the object then naturally occupies a larger portion of the image. However, only the part of the object close to the center of the image will be in line with the camera model.

4.5.5 Future work

Improvements for the plane exploitation feature can be performed which removes the restrictions made on the object. The implementation can be extended to allow for flat parts on the object which spans a plane that is not axis orthogonal. Furthermore, in addition to the predefined lying positions, a user could be able to specify the set of points on the CAD-model that are known to be attached to the plane, similar to the feature of adding new keypoints.

The problem of the pinhole camera model bias could be partially solved by allowing users to provide distortion models in conjunction with the inner parameters, such as the radial distortion model [56] or the division model [57]. This would allow for the tool to use the corresponding correction model (inverse distortion models) to obtain corrected 2D keypoint coordinates better suited for the pose reconstruction. Alternatively, since the pinhole camera model is very commonly used, and users may not have better models to provide, the tool could be extended to simply warn users when localizing 2D keypoints in further away from the image center, where the pinhole model is typically less accurate. Yet another alternative could be to introduce an uncertainty associated with the distance from the image center for the 2D points, and perform weighted optimization. The scaling of the uncertainty levels would have to be chosen by the user, depending on how accurately they believe their pinhole camera model models the real camera.

In order to let the user more efficiently navigate the viewpoints, the tool could have exploited the spatial information contained in the motion object. An example would be to add a navigation button in the main view which skips to the "next-best-view" for keypoint identification. This could e.g. be the next viewpoint in the sequence where the camera matrix differs enough from the prior camera matrix. Intuitively, identifying 2D keypoints from significantly differing viewpoints should lead to better pose estimates [58] than doing so from closely adjacent cameras. Rather than having the user navigate to such viewpoints, such a system could potentially do it for them in many cases.

Finally, as discussed above, we recognize that our Annotation Tool currently assumes that the user is interested in finding 6DoF poses for any object class. This does not have to be the case, and functionality which takes this into account should be

considered for future development of the tool. Such funtionality would appropriately be realized with separate optimizers for objects indicated by the user as rotationally symmetric.

5

Detection Algorithm

This chapter describes the whole pipeline that was used for global detection of 6D poses, presents the results and discusses them.

Given a sequence $\{I_t\}_{t=1}^{T}$ of T RGB images with an associated camera motion $\{P_t\}_{t=1}^{T}$ and the inner parameters K, the goal of the algorithm is to detect instances of specific inventory objects, and estimate their 6D poses in a global coordinate system. Note that the detections are not accompanied by confidence scores, disqualifying the results to be evaluated with the standard metric mAP.

5.1 Method

In order to achieve the task at hand, we have designed a pipeline which for each image I_t in the sequence calculates an output independently. It then aggregates the outputs to produce the final object detections and pose estimations, utilizing the motion $\{P_t\}_{t=1}^{T}$. The single-view procedure heavily makes use of modules developed and published by S. Peng et al. for their work PVNet [20]. This choice is motivated in Section 3.1. A flowchart for the pipeline as a whole is shown in Figure 5.1. We justify the choice of processing each image in the sequence independently by our assumption of static scenes; Taking the sequential nature of the images into account should yield no additional information to exploit. Furthermore, for reasons stated below in Section 5.1.1, the entire pipeline is built to handle one object class at a time.

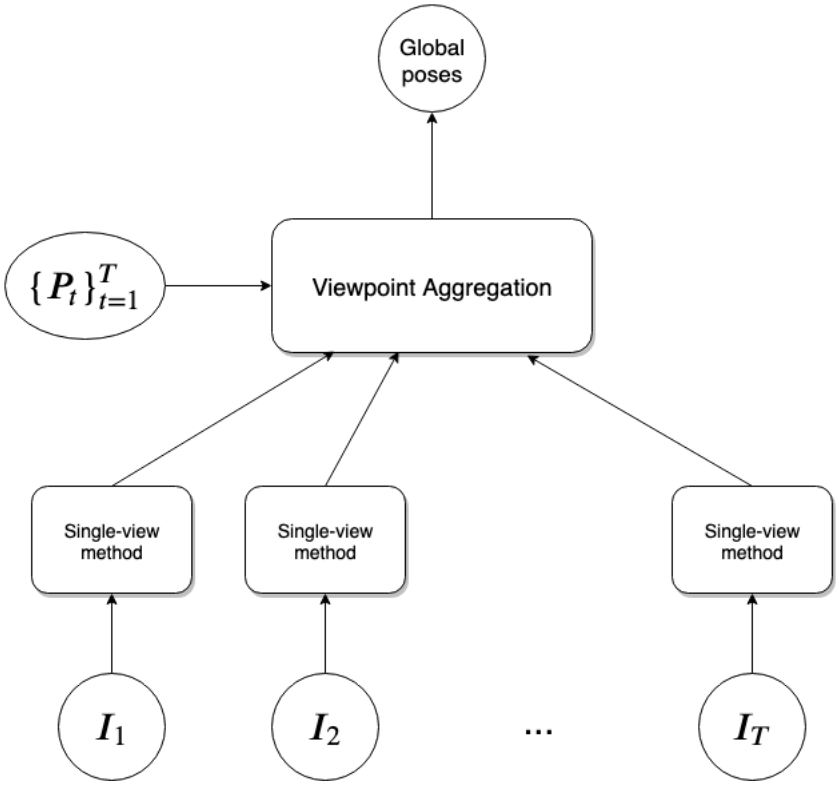


Figure 5.1: Overview of the Detection Algorithm. The algorithm first processes each image of the scene independently to find quantities that may be aggregated simultaneously inthe final step, producing the detections and pose estimates in the scene.

The resulting Detection Algorithm was evaluated by calculating the F_1-score as described in Section 2.3, on each of our datasets, both the validation sets and the training sets. The results reported on the validation indicates how well the algorithm performs on unseen data, i.e. how well it generalizes. An estimated pose was considered a match with a ground truth pose if the ADD metric between them was less than 10% of the object diameter, and the Hungarian Algorithm was used for assignment between estimated and ground truth poses. We claim there is value in evaluating the pipeline even on the training datasets, since these results provide insight in how well the non-NN components of the algorithm perform specifically, and let us assess how sensitive the pipeline is to good regressions from the NN (assuming the NN performs better on the training data to which it is fitted). Furthermore, since the pipeline makes use of a deep learning module, the final performance metrics (loss function values) on both the training and validation set of the module are presented.

In this section, we describe our complete pipeline for estimating global poses through the following steps: First we present an exhaustive review of the PVNet [20] method for single-view pose estimation. Since our single-view procedure uses modules from that work, we present their methods separately to make clear what parts of our algorithm stem from their contributions. Secondly, our single-view procedure, the

output of which is used in the aggregation step, is explained in detail. Finally, we explain how the data extracted from each image by the single-view procedure is aggregated to yield the final object detections and pose estimates.

5.1.1 Single-view pose estimation using PVNet

The PVNet architecture is designed to detect an instance of a specific object, and estimate its pose relative the camera, assuming there is at most one instance of the object present in the image. The inability to process images with multiple instances of the object makes the method unsuitable to include as it is in our pipeline. However, parts of their methods are directly applicable to the task our single-view component of the pipeline aims to carry out. Therefore, this section aims to explain these parts.

Similar to many other recent works within single-view pose estimation, PVNet breaks down the problem in two main steps consisting of using a deep learning based approach for creating a set of 2D-3D correspondences and solving the PnP problem defined in Section 2.1.2.2. The core idea of PVNet is to regress a set of 2D vectors for each object pixel that points towards the locations of a set of keypoints in the image.

The following are the key components of the PVNet pipeline:

- A NN is trained to simultaneously perform class segmentation, i.e. predict which class a pixels belong to, and within the segmented area predict the voting direction to the set of keypoints for the object.
- A RANSAC scheme is used to generate a number of hypotheses for the location of keypoints, coupled with a score for the "goodness" of the hypothesis.
- The hypotheses and their scores are then used to estimate the keypoints and their covariances in a probability distribution fashion.
- A modified version of the PnP-problem is then solved for, which utilize the estimated uncertainties in the keypoints.

Although the NN can be used to predict the desired outputs for all classes simultaneously, an alternative implementation is to train a separate NN for each class separately. As far as the authors are aware, the latter case is the one adopted by the PVNet authors. This is also the approach we have used, as it allows for new object types to be incorporated to the problem without having to retrain the NN each time we do so. This means that the detection of the objects are done separately between the classes during the whole pipeline. Following sections assume this implementation.

5.1.1.1 Pixel-wise voting and class segmentation

As explained in Section 2.1.6, a set of keypoints $\{\boldsymbol{X}_k\}_{k=1}^K$ belonging to a known model of an object may be defined. By detecting the projection $\{\boldsymbol{x}_k\}_{k=1}^K$ of these keypoints in an image, and thus acquiring a set of 2D-3D correspondences, solving the PnP problem would provide the the single-view pose of the object in an image.

The idea behind the PVNet method is to infer $\{\boldsymbol{x}_k\}_{k=1}^K$, which may be occluded in the image or outside the image border, from visible parts of the object. This is done by predicting the direction from each of the *object pixels* (i.e. pixels belonging to visible parts of the object) towards $\{\boldsymbol{x}_k\}_{k=1}^K$.

Consider the coordinate $\boldsymbol{p}$ of a pixel in the image and the coordinate for a single projected keypoint $\boldsymbol{x}_k$ in the image. The direction from $\boldsymbol{p}$ towards $\boldsymbol{x}_k$ is defined by the *voting vector*

$$\boldsymbol{v}_k(\boldsymbol{p}) = \frac{\boldsymbol{x}_k - \boldsymbol{p}}{||\boldsymbol{x}_k - \boldsymbol{p}||_2} \tag{5.1}$$

which is the main quantity of interest in the PVNet method. The prediction of $\boldsymbol{v}_k(\boldsymbol{p})$ is made for each of the pixels in the image and for each keypoint meaning that the complete quantity is a tensor of dimension $(H \times W \times (K \times 2))$, where H and W is the height and width of the image. We refer to this quantity as the *voting field*, and will interchangeably use this name to reference both the complete quantity and a subset of it for a single keypoint.

To make the prediction, a CNN is used as explained in Section 5.1.1.2 below. Although the prediction is made for all pixels, only the object pixels are used in the following steps of the algorithm. This is reasonable since CNN's makes use of local features in an image. PVNet is therefore designed to also predict object pixels, i.e. perform semantic segmentation, which is represented by a tensor of dimension $(H \times W \times 2)$. The segmentation is depicted in Figure 5.3 and the voting field of a single keypoint within the segmented area is visualized in Figure 5.2.

5.1.1.2 Neural network design

The NN of PVNet was implemented in the PyTorch computing framework for machine learning [59]. The input to the NN consists solely of an RGB image, which is a tensor of dimensions $(H \times W \times 3)$. The output is a tensor containing both the voting vector field and the segmentation predictions stacked along the third dimension, i.e. it is of dimension $(H \times W \times (K \times 2 + 2))$. The neural network model is a revised version of a pretrained ResNet-18 network [60], with the following alterations:

- The pooling layers following the feature maps of dimension $\left(\frac{H}{8} \times \frac{W}{8}\right)$ are discarded.
- The subsequent convolutions are replaced by dilated convolutions.

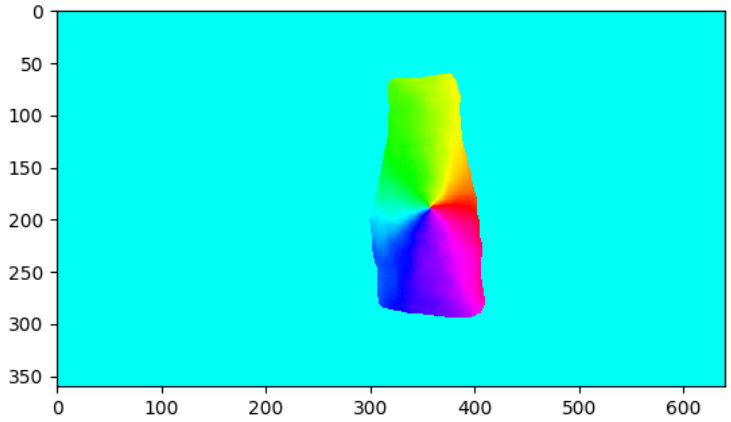


Figure 5.2: Example of a voting field corresponding to a keypoint. Each pixel in the semantically segmented area of the image has a direction $\boldsymbol{v}_k$ pointing towards keypoint k. The directions are color-encoded in this illustration.

Figure 5.3: Example of semantic segmentation. A semantic mask (transparent green) of all pixels belonging to the blue soap bottle covers the bottle in the image.

- Skip connections, convolution and upsampling is added after the feature map until it reaches the dimensions $(H \times W)$ and finally a (1×1) convolution layer is applied.

Note that the output voting field during testing might not be vectors of unit length, and are therefore normalized after the prediction.

The network, parameterized by the weights $\boldsymbol{w}$, is trained with a separate loss function for the segmentation and the voting field. The loss function l_v for the voting field is defined for a single image by

$$l_v(\boldsymbol{w}) = \sum_{k=1}^{K} \sum_{\boldsymbol{p} \in O} l_1(\Delta \boldsymbol{v}_k(\boldsymbol{p}; \boldsymbol{w})|_x) + l_1(\Delta \boldsymbol{v}_k(\boldsymbol{p}; \boldsymbol{w})|_y)$$
$$\Delta \boldsymbol{v}_k(\boldsymbol{p}; \boldsymbol{w}) = \tilde{\boldsymbol{v}_k}(\boldsymbol{p}; \boldsymbol{w}) - \boldsymbol{v}_k(\boldsymbol{p})$$

where $\tilde{\boldsymbol{v}}_k$ is the predicted voting vector, while $\boldsymbol{v}_k$ is the ground truth voting vector and the two elements $\boldsymbol{v}_k|_x$ and $\boldsymbol{v}_k|_y$ are the two elements of $\boldsymbol{v}_k$. O is the set of pixels belonging to the ground truth segmentation and $\boldsymbol{w}$ is the parameters of the network. $l_1(x)$ is the *smooth* l_1 *loss* as defined by

$$l_1(x) = \begin{cases} 0.5x^2 \text{ if } |x| < 1 \\ |x| - 0.5 \text{ otherwise}. \end{cases}$$

The segmentation loss l_s (5.2) is the mean cross-entropy as described in Section 2.2 over all $P = H \cdot W$ pixels in the image. Here, $y \in \{0, 1\}$ is the target class of a pixel, and $\tilde{y}$ is the predicted output.

$$l_s(\boldsymbol{w}) = \frac{1}{P} \sum_{i=1}^{P} y^{(i)} \cdot \log\left(\tilde{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \cdot \log\left(1 - \tilde{y}^{(i)}\right) \tag{5.2}$$

The complete loss is then taken as a sum of the two $l = l_v + l_s$. The total loss for the dataset is the mean loss over all images in the dataset.

5.1.1.3 Hypothesis generation and keypoint estimation

The next step in the pipeline is estimating the keypoints. The PVNet pipeline determines an uncertainty measure for the the keypoints it detects and model the keypoints by a Gaussian probability distribution.

First a number of N keypoint hypotheses $\{\boldsymbol{h}_{k,i} | i = 1, 2, ..., N\}$ are calculated for each keypoint type. This is done by sampling pairs of random pixels within the segmented area and taking the intersection of the two *voting lines* which are formed by extending their voting vectors. Each of the hypotheses are then scored according

to how many pixels vote for them according to

$$w_{k,i} = \sum_{\boldsymbol{p} \in O} \mathbb{I}\left(\frac{(\boldsymbol{h}_{k,i} - \boldsymbol{p})^T}{||\boldsymbol{h}_{k,i} - \boldsymbol{p}||_2} \tilde{\boldsymbol{v}_k}(\boldsymbol{p}) \geq \theta \right) \tag{5.3}$$

where $\mathbb{I}$ is the indicator function, θ is a threshold parameter and $\boldsymbol{p} \in O$ denotes the set of predicted object pixels. We denote the composite function in the summation the *inlier function* for future reference. The resulting scores of equation (5.3) are used to estimate the mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$ for a keypoint $\boldsymbol{x}_k$ according to equations (5.4) and (5.5).

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} w_{k,i} \boldsymbol{h}_{k,i}}{\sum_{i=1}^{N} w_{k,i}} \tag{5.4}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{N} w_{k,i} (\boldsymbol{h}_{k,i} - \boldsymbol{\mu}_k)(\boldsymbol{h}_{k,i} - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^{N} w_{k,i}} \tag{5.5}$$

5.1.1.4 Uncertainty-driven PnP

To exploit the probability distribution representation of $\boldsymbol{x}_k$, PVNet solves the PnP problem by minimizing the Mahalanobis distance instead of the reprojection error defined in equation 2.12. The minimization problem is then

$$\min_{R,\boldsymbol{t}} \sum_{k=1}^{K} \boldsymbol{r}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{r}_k \tag{5.6}$$

$$\boldsymbol{r}_k = \pi(R\boldsymbol{X}_k + \boldsymbol{t}) - \boldsymbol{x}_k \,. \tag{5.7}$$

This problem is denoted *Uncertainty-driven PnP*, for which the solution will provide the pose $P = [R, t]$.

5.1.2 Our single-view pipeline

Our single-view procedure aims to, given a single RGB image of the scene, produce output containing information which can be aggregated to yield accurate object detections and pose estimations. The output this procedure aims to produce is for each instance visible in the image, all K corresponding 2D keypoint locations $\{\boldsymbol{x}_k\}_{k=1}^{K}$. The choice of this output quantity, rather than alternatives such as direct pose estimates, provides interpretability in the intermediate steps of the pipeline, while also allowing for flexibility in the aggregation step.

To calculate this output, the method needs a way of deciding upon a number of instances visible in the image, and regress the 2D keypoint locations for each instance. Briefly put, this method aims to separate the semantic segmentations produced by the PVNet neural network into instance segmentations, and thereafter

use the RANSAC procedure (see Section 5.1.1.3) to find the 2D keypoint locations for each separate instance using the split segmentations. In the original paper on PVNet, the authors proposed a simple method to separate the semantic segmentations (for which no implementation was published), but the method was deemed to be inadequate for its purpose when we tested it. An overview of the single-view procedure is presented in Figure 5.4.

The pipeline starts with the use of a Neural Network for predicting the voting field and the semantic segmentation in a single image. These two quantities are used to generate a number of N keypoint hypotheses for each keypoint type. Multiple keypoints are then selected from the hypotheses by using a clustering method. The instance segmentations are determined by first deciding upon the number S of visible objects and then grouping pixels according to similarity in voting patterns. Finally the sets of 2D keypoints for each of the S instances are determined by running the PVNet pipeline from the hypotheses generation step for each of the instances segmented areas individually.

5.1.2.1 Neural network

To predict the voting field and semantic segmentation we use the same network architecture as the work of PVNet, described in Section 5.1.1.2. When training the network we also used the same loss function and hyperparameters.

As explained in Section 5.1.1, a neural network was trained for each of the different objects in our inventory. To produce the labels for training we made use of the annotated poses. The semantic segmentation labels were acquired by using external software which rendered the models according to the poses. The voting fields were calculated by using the pose data to project the model keypoints to the image and then calculating the voting vectors according to equation (5.1).

5.1.2.2 Hypothesis generation

Given the output from the Neural Network, we follow the procedure for acquiring a number of hypotheses $\{\boldsymbol{h}_{k,i} | i = 1, 2, ..., N\}$ in the exact same way as outlined in Section 5.1.1.3. To score the hypotheses, we do however use an alternative to the inlier function in equation 5.3 which we denote a *voting function* $V_a(\boldsymbol{h}, \boldsymbol{p}, \boldsymbol{v})$:

$$V_a(\boldsymbol{h}, \boldsymbol{p}, \tilde{\boldsymbol{v}}) = \exp(-aL^2) \cdot \max\left\{\frac{(\boldsymbol{h}-\boldsymbol{p})^T\tilde{\boldsymbol{v}}}{||\boldsymbol{h}-\boldsymbol{p}||_2}, 0\right\} \tag{5.8}$$

$$L^2 = ||\boldsymbol{h}-\boldsymbol{p}||_2^2 - \left((\boldsymbol{h}-\boldsymbol{p})^T\tilde{\boldsymbol{v}}\right)^2, \tag{5.9}$$

where L is the orthogonal distance between the voting line and the hypothesis, and a is a parameter of the function. The reasons for using the voting function rather than the inlier function is to get a continuous value in $(0, 1)$ of "how much" a pixel votes for a hypothesis. The need for this will be apparent in Section 5.1.2.5.

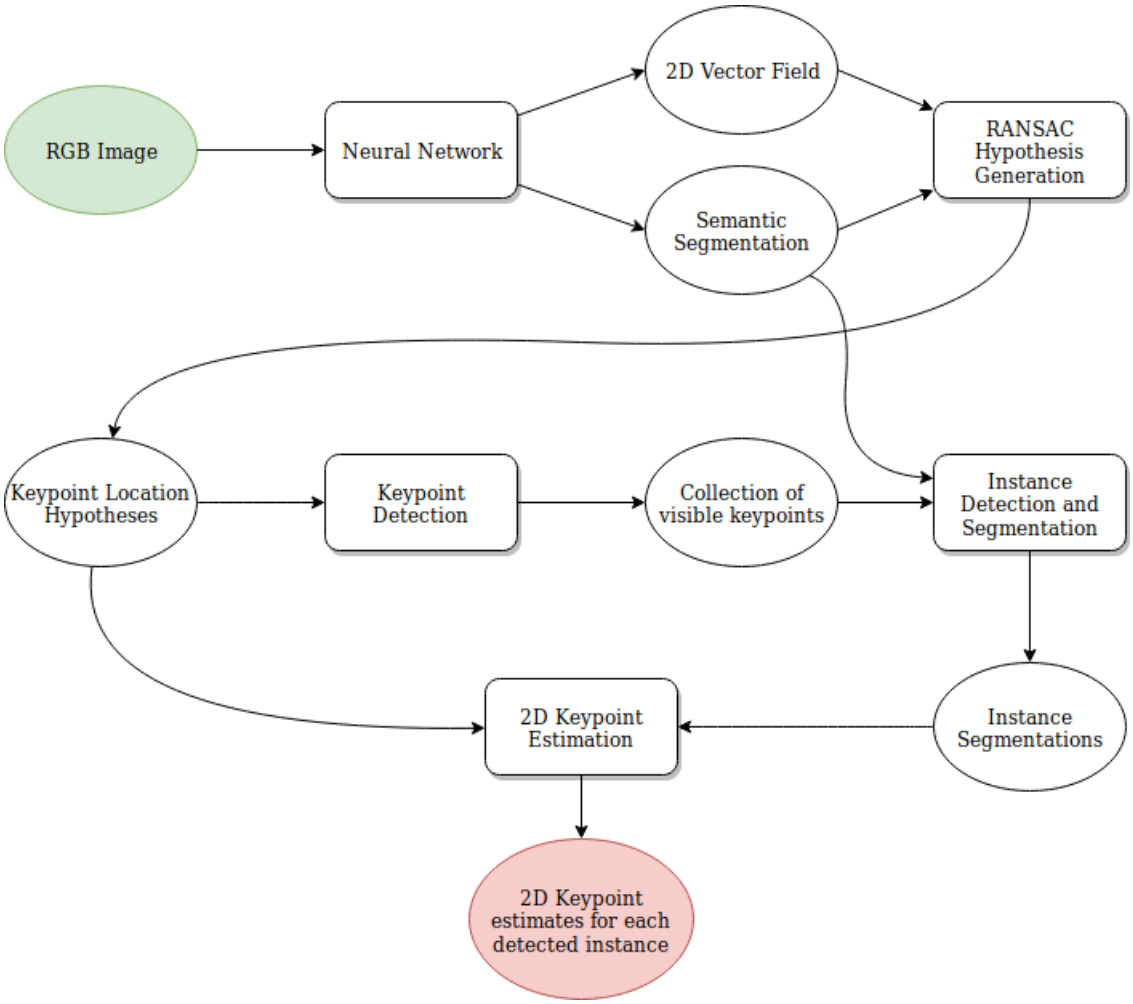


Figure 5.4: Flowchart describing the various components making up the single-view pipeline, which takes one RGB image and outputs estimates of the 2D keypoint locations of object instances detected in the image.

The form of V_a consisting of the two terms $\exp(-aL^2)$ and $\max\{\frac{(\boldsymbol{h}-\boldsymbol{p})^T\boldsymbol{v}}{||\boldsymbol{h}-\boldsymbol{p}||_2}, 0\}$ is motivated by the requirement that a pixel both vote spatially close to the hypothesis as well as in the correct direction to get a high voting score. For instance, a pixel which is placed at great distance from the hypothesis may get a high value of the term $\max\{\frac{(\boldsymbol{h}-\boldsymbol{p})^T\boldsymbol{v}}{||\boldsymbol{h}-\boldsymbol{p}||_2}, 0\}$ even though it votes at a large distance L from the hypothesis. On the other hand, a pixel placed really close to the hypothesis will get a large value of the term $\exp(-aL^2)$ since L will be small even though it might vote in a quite different direction than the hypothesis location. Using both these terms balances out these two extreme cases and worked overall better for our purposes. Using a larger value of the parameter a balances the score to prioritize the spatial measure.

5.1.2.3 Keypoint detection

In the case of single instances in an image, the generated hypotheses data is interpreted as being drawn from a Gaussian probability distribution as noted in Section 5.1.1.3, for which we associate a mean/mode μ and a covariance Σ. In the case of multiple instances of a keypoint type in an image, the distribution becomes a mixture model, and we aim to find the all modes of the distribution. The mode detection is accomplished by using Mean-Shift clustering (MSC), as it was designed for this. Although MSC can be performed with the hypotheses locations only, we propose to incorporate another quantity in the data.

We observe a pattern that the orientation of a particular hypotheses cluster, i.e. the principal components of the data in the cluster, seem to be aligned with the relative direction between the cluster and the object pixels as seen in Figure 5.5. We may therefore expect that the hypotheses in this cluster on average get their votes from approximately the same direction as the principal components. Motivated by this observation we define a additional property for a hypothesis $\boldsymbol{h}_i$ which we denote the *voting direction* $\boldsymbol{d}_i$

$$\boldsymbol{d}_i = \frac{\sum_{\boldsymbol{p}\in O} \frac{(\boldsymbol{h}_i-\boldsymbol{p}_j)}{||(\boldsymbol{h}_i-\boldsymbol{p}_j)||_2} \cdot V_a(\boldsymbol{h}_i, \boldsymbol{p}_j, \boldsymbol{v}_j)}{\sum_{\boldsymbol{p}\in O} V_a(\boldsymbol{h}_i, \boldsymbol{p}_j, \boldsymbol{v}_j)} \,. \tag{5.10}$$

This property represents which direction a hypothesis got its votes from on average and approximates the principal components of the cluster it belongs to. The idea is to exploit this property by performing MSC in a higher dimensional space where each hypothesis point is represented not only by its location $\boldsymbol{x}$, but also its voting direction $\boldsymbol{d}$. This is advantageous over just clustering on the locations when the locations of the keypoints are near each other, but they steam from the votes from different instances. A sparse set of voting directions can be seen in Figure 5.6.

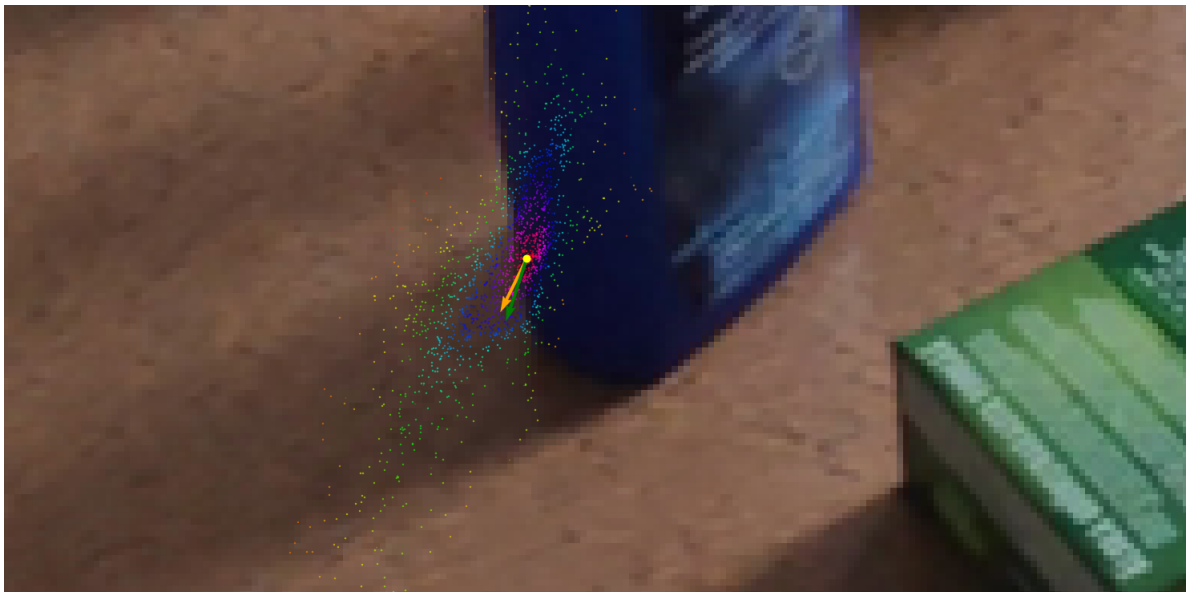


Figure 5.5: The voting direction (orange arrow) of the MSC detected keypoint (yellow point) and the principal component (green arrow) starting at the keypoint plotted among the keypoint hypotheses. The keypoints hypotheses are color coded according to voting score.

Figure 5.6: Voting directions for a sparse set of keypoint hypotheses. The orange arrows indicate the average voting direction from all pixels $\boldsymbol{p} \in O$. The yellow circles are examples of keypoints detected after MSC in the combined space of location and voting directions. The keypoints hypotheses are color coded according to voting score.

5.1.2.4 Instance detection and segmentation

At this point we have acquired a set of keypoints for each keypoint type and aim to use this data to first determine the number S of visible instances in the image and then split the semantic segmentations into S instance segmentations.

To estimate S we resort to looking at the number of detected keypoints for each keypoint type. S is taken to be the most common number of detections among the different keypoint types.

The main idea for grouping the pixels in S categories is that pixels belonging to the same instance are expected to vote similarly across all detected keypoints for all keypoint types. To exploit this assumption we make use of the voting function V_a defined in equation (5.8). The complete set of votes for a particular pixel can be represented as a point in $\mathbb{R}^{K_d}$, where K_d is the total number of detected keypoints across all keypoint types. Due to our choice of V_a, the votes will have values in a continuous range in contrast to the indicator function. As pixels are assumed to vote similarly, they are expected to be spatially close in this space. We therefore make use of K-Means clustering with S centroids to group the pixels. Note that the reason for defining an alternate voting function was due to the requirement of acquiring voting scores in a continuous range for clustering purposes.

5.1.2.5 Instance 2D keypoint estimation

Given the split segmentations from the previous step, the original PVNet pipeline is now used with each instance of the segmentations separately. This provides S sets of all K keypoints for the visible instances which is desired data from each image.

5.1.3 Aggregation

Given a set of keypoints for each instance in all images we now aim to aggregate these into the global pose estimates, and decide how many instances of the class are present in the scene. From viewpoint t, a collection of i_t sets of 2D keypoint locations in that image is given. The overall collection of keypoints is assumed to contain noisy data, and outliers.

Since the keypoint locations were calculated independently for each image by the single-view procedure, the indices of detected instances in each viewpoint are not consistent with indices in other viewpoints. For example, given 2 sets of keypoints from image I_1 and 2 sets of keypoints in image I_2, we have no way of knowing if instance 1 detected in I_1 corresponds to instance 1 or 2 in I_2. See Figure 5.7 for a visualization of the scenario. Therefore, the first step of our aggregation method is to match the instances detected in the various images. This will subsequently yield

Figure 5.7: Point estimates from single-view procedure for two different viewpoints of the same scene. In the viewpoint 371, the leftmost box of macaroni was detected as the first instance (blue dots), whereas in viewpoint 347, the rightmost was.

the final number of detections of an object class in the scene. After that, the pose of each object instance is obtained using Multi-View PnP.

The first step taken is to simply fit a pose to each of the sets $\{\boldsymbol{x}_k\}_{k=1}^{K}$ across all T images, and from the poses express the object centers in a unified global coordinate system. The idea is that if sufficiently accurate 2D keypoint locations were regressed from enough images I_t, the object centers in $\mathbb{R}^3$ will cluster around the true object center point.

The next step is to detect these clusters. Once the clusters have been detected, instances in different viewpoints can be matched; 2D keypoints which generated object center hypotheses which belong to the same cluster are considered to belong to the same object instance. Note that clustering methods do not necessarily assign every data point to a cluster. 2D keypoints corresponding to such data points are considered outliers, and are filtered out.

The clustering method used to find the clusters was Mean-shift clustering with a parameter sweep. Since the final number of detected instances is decided upon in this step, it is crucial for the clustering method to find the correct number of clusters. To make the method more robust in this regard, the bandwidth parameter h is swept over the interval $[\frac{1}{4}D_{min}, D_{min}]$ where D_{min} is the minimum point cloud axis limit. The reasoning behind these limits stems from the physical interpretation of the bandwidth parameter of the Mean-shift kernel; The center points of two clusters are not expected to end up within a radius h. Therefore, we set the upper limit of the h to the closest distance the center points of two object of the same class may have. After having performed Mean-shift clustering sweep, a set of cluster centers is obtained for every parameter value. To find the final number of clusters, a single Mean-shift clustering procedure is performed on these cluster centers. This will detect the center points of the cluster center data which will yield one hypothetical center point per detected instance in the scene.

Now, a set of I global instance centers have been detected, and the image-specific instances are to be matched by assignation to one or none of the global instances. For each image-specific instance, the distances between the center point of the pose fitted to the corresponding 2D keypoints, and each of the global instance centers are calculated. The image-specific instance is then either assigned to the global instance for which the distance is smallest, or to no instance if the smallest distance is above a certain threshold. Now, for each global instance $i = 1, 2, ...I$, there is a collection $\{\boldsymbol{x}_{t,k}\}$ for some $t \in \{1, 2, ..., T\}$ and all $k = 1, 2, ...K$. Finally, to obtain the poses of the detected instances, Multi-view PnP is carried out on that data. Finally, the algorithm filters our duplicates among its detections by deleting instances which are too similar. An instance is considered a duplicate if the ADD-metric between it and another instance's pose is below 10% of the object diameter (the same criterion as used for instance matching during evaluation).

5.2 Results

Table 5.1: F_1-score over all scenes in the validation set for each class.

Object Class	F_1-score	Precision	Recall
Deciliter Measure	0.3750	0.2857	0.5455
Coconut Milk	0.4000	0.5000	0.3333
Paper Cup	0.1739	0.1667	0.1818
Macaroni	0.7142	0.7692	0.6666
Soap (Green)	0.6666	0.6154	0.7273
Soap (Blue)	0.3157	0.3000	0.3333
Lentils	0.2399	0.2500	0.2308

Table 5.2: F_1-score over all classes for each scene in the validation set.

Scene Idx	F_1-score	Difficulty
1	0.6229	Medium
2	0.3333	Hard
3	0.2951	Hard

Table 5.3: F_1-score over all scenes in the training set for each class.

Object Class	F_1-score	Precision	Recall
Deciliter Measure	0.9037	0.8592	0.9531
Coconut Milk	0.6839	0.7910	0.6023
Paper Cup	0.6168	0.7174	0.5410
Macaroni	0.7625	0.8413	0.6974
Soap (Green)	0.7500	0.6957	0.8136
Soap (Blue)	0.8888	0.8955	0.8823
Lentils	0.6861	0.7966	0.6026

Table 5.4: F_1-score over all classes for each scene in the training set.

Scene Idx	F_1-score	Difficulty
1	1.00	Easy
2	0.7576	Hard
3	0.6170	Hard
4	1.00	Easy
5	0.8667	Easy
6	0.4810	Hard
7	0.8511	Medium
8	0.8800	Easy
9	0.6000	Hard
10	0.5283	Hard
11	0.8136	Hard
12	0.9412	Easy
13	1.00	Medium
14	0.5217	Medium
15	0.8500	Medium
16	0.7143	Medium
17	0.8247	Medium
18	1.00	Easy
19	0.7883	Hard

Table 5.5: Table of segmentation loss, voting loss and total loss values on training and validation data for the different classes. The classes are indexed as in Table 3.3.

	1	2	3	4	5	6	7
Train loss (Total)	0.0063	0.0073	0.0077	0.0081	0.0065	0.0043	0.0058
Train loss (Voting)	0.0102	0.0114	0.0118	0.0102	0.0094	0.0060	0.0082
Train loss (Segmentation)	0.0024	0.0031	0.0037	0.0060	0.0037	0.0026	0.0034
Validation loss (Total)	0.0248	0.0373	0.0132	0.0297	0.0204	0.0212	0.0366
Validation loss (Voting)	0.0315	0.0520	0.0200	0.0369	0.0251	0.0176	0.0350
Validation Loss (Segmentation)	0.0182	0.0227	0.0065	0.0225	0.0158	0.0249	0.0381

5.3 Discussion

The results in Table 5.1 shows that our algorithm performs very poorly on new data. The class for which it detected objects with the best F_1-score was the Macaroni Box class, and for that class the score was 0.7142, significantly below the desired value of 1. Furthermore, the results on the training dataset are not impressive either, where the algorithm scored 0.9037 for the best class. We believe that the poor results stems partly from questionable design choices in our algorithm, and partly from poor training of the NN module. Both aspects are discussed below, with suggestions on how the algorithm could have been improved.

5.3.1 Design choices and potential improvements

On the highest level, we considered the idea of solving the problem by determining keypoints in multiple images and using a multi-view PnP approach for aggregated pose estimation to be well grounded. For instance, this is the basic idea that the annotation tool was based on and multiple works of others have shown success in keypoint detection. Our solution for tackling the two major problems of keypoint detection in an image and keypoint matching between images was however insufficient.

Although the choice of the PVNet pipeline for keypoint detection was well motivated for a number of reasons, we did not consider differences in the data we would work with. During the development of our algorithm, we discovered scenarios ex-

hibited in our data which the PVNet method was unsuited to handle. PVNet was mainly designed for detecting single instances in an image, and essentially solves the localization problem, which did not translate well to the multi-instance case. This may be most evident from the noisy voting field in images where there was heavy occlusion between objects of the same class. See Figure A.1 in Appendix A for a comparison between the ground truth voting field, predicted voting field and the resulting hypotheses in such a case.

Furthermore, in these cases, keypoints of the same type tend to be spatially close in the image which results in hypotheses clusters overlapping. The authors proposed a method for extending the PVNet method for detection of multiple instances and keypoints of the same type, but it did not suffice in the scenario of same class self-occlusion. Their proposed method seemingly assumed that multiple instances of a class were located far away from each other. Consequently, the predicted number of instances and the segmentation was often incorrect, which may be an argument against the fundamental idea of a voting field in this particular scenario.

The issues of determining the number of instances and segmentation in cases of same class self-occlusion propagated into the stage of keypoint matching between images. Since our method for keypoint matching made use of single-view pose estimation, it required matched keypoint sets within the image to be determined. As a consequence, much effort and time was spent designing more sophisticated procedures which could handle the problems that were due to same class self-occlusion.

A better alternative for the keypoint matching could have been to examine solutions where the poses were not needed for aggregation of keypoint detections. For instance, the unmatched keypoints in an image could have been used with a triangulation approach to the problem. This would provide a set of unmatched global 3D points of different keypoint types for which the poses could have been solved for. This would also have the advantage of exploiting images with a single or few detected keypoints, where pose predictions are unfeasible. In hindsight, such a method would have been more justifiable since we aimed to solve the problem with heavy occlusion.

Although we are critical of our choices for keypoint detection and keypoint matching, we consider a large part of our insufficient result to stem from too little effort invested in the neural network training stage. The inadequate performance of the NN is evident from both the losses in Table 5.5 and the final F1-scores for the training versus the validation datasets. In addition we have the previously mentioned qualitative observation of the voting field performance in the image A.1 presented in Appendix A. We believe than an improvement could have been made by more experiments regarding the hyper parameters of the network. Furthermore, online data augmentation could have been implemented which has proven to be an effective tool for training purposes. Since our training set consisted of merely 8208 images, we question whether the amount of data was sufficient for training and data augmentation would in this case be a particular good idea.

6

Conclusions

For this project, we have developed a tool for 6D pose annotation of RGB images which we used to create a new dataset for object detection and 6D pose estimation in a supermarket setting. We have also developed an algorithm for performing this task, and evaluated it on our dataset.

The Annotation Tool lets a user identify locations of keypoints belonging to objects from a few viewpoints, and uses this data to find the pose of the object relative all images. We believe that the final product facilitates efficient annotation of large datasets, and that the interface design and additional features integrated in the tool help reduce the workload. However, in its current state, the tool needs the user to exercise caution with respect to potentially insufficiently accurate camera modeling, and pose fitting of rotationally symmetric objects. The latter aspect may have affected the quality of the labels of some objects in our dataset negatively. Future development of the tool should thus focus on expanding the tool with alternative camera models and optimizers which allow for poses with fewer degrees of freedom.

Our Detection Algorithm incorporated parts of a single-view pose estimation method to produce sets of object instance keypoints, and aggregated that data from several viewpoints by fitting poses and clustering the pose centers using a Mean-Shift clustering procedure. From our results, both from evaluation on unseen and previously seen data, we conclude that our method is not suitable for the task of visual inventory. The performance was especially poor in scenes which exhibited great levels of occlusions, as one would expect of the target environment of supermarket shelves. We believe that our intermediate goal of producing instance-matched keypoint detections in every image was misguided, as the deep learning model used produced output data which, in images exhibiting occlusions from instances of the same class, made this task difficult. Furthermore, the deep learning module failed to produce accurate output in these scenarios, potentially due to lack of training data. For future work using an algorithm structure similar to ours, we recommend either replacing the deep learning module, or changing the target output of the single-view procedure and consequently exploring alternative methods of data aggregation for pose detection.

Bibliography

[1] "Semantic mapping and visual navigation for smart robots." https://www.chalmers.se/en/projects/Pages/Semantisk-kartering-Q-visuell-navigering-fQr-smarta-robotar.aspx. Accessed: 2019-01-21.

[2] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," in *Proc. Asian Conf. Computer Vision*, vol. 7724, jan 2012.

[3] T.-T. Do, M. Cai, T. Pham, and I. Reid, "Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image," 2018.

[4] O. H. Jafari, S. K. Mustikovela, K. Pertsch, E. Brachmann, and C. Rother, "The best of both worlds: Learning geometry-based 6d object pose estimation," *CoRR*, vol. abs/1712.01924, 2017.

[5] A. Zeng, K. Yu, S. Song, D. Suo, E. W. Jr., A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," *CoRR*, vol. abs/1609.09475, 2016.

[6] J. Sock, S. H. Kasaei, and L. S. Lopes, "Multi-view 6D Object Pose Estimation and Camera Motion Planning using RGBD Images," tech. rep.

[7] C. Li, J. Bai, and G. D. Hager, "A Unified Framework for Multi-View Multi-Class Object Pose Estimation," mar 2018.

[8] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim, "Latent-class hough forests for 3d object detection and pose estimation," in *European Conference on Computer Vision*, pp. 462–477, Springer, 2014.

[9] 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.

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

[11] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 683–698, 2018.

[12] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3d orientation learning for 6d object detection from rgb images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 699–715, 2018.

[13] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[15] Zhe Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6dof pose estimation for textureless objects," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2441–2448, May 2016.

[16] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of textureless objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 876–888, May 2012.

[17] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3d object modeling and recognition using affine-invariant patches and multi-view spatial constraints," vol. 2, pp. II– 272, 07 2003.

[18] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, (Washington, DC, USA), pp. 1150–, IEEE Computer Society, 1999.

[19] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3d pose estimation," *CoRR*, vol. abs/1502.05908, 2015.

[20] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "Pvnet: Pixel-wise voting network for 6dof pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4561–4570, 2019.

[21] S. L. Chiu, "Fuzzy model identification based on cluster estimation," *J. Intell. Fuzzy Syst.*, vol. 2, pp. 267–278, May 1994.

[22] A. Aldoma, F. Tombari, L. Di Stefano, and M. Vincze, "A Global Hypotheses Verification Method for 3D Object Recognition," in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), (Berlin, Heidelberg), pp. 511–524, Springer Berlin Heidelberg, 2012.

[23] F. Viksten, R. Soderberg, K. Nordberg, and C. Perwass, "Increasing pose estimation performance using multi-cue integration," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 3760–3767, May 2006.

[24] T. Volodine, "Point cloud processing using linear algebra and graph theory," 2007.

[25] G. Malandain and J.-D. Boissonnat, "Computing the diameter of a point set," *International Journal of Computational Geometry & Applications*, vol. 12, no. 06, pp. 489–509, 2002.

[26] V. Lepetit and P. Fua, "Keypoint recognition using randomized trees," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 9, pp. 1465–1479, 2006.

[27] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-dof object pose from semantic keypoints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2011–2018, IEEE, 2017.

[28] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7074–7082, 2017.

[29] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learining Internal Representations by Error Propagation," 1986.

[31] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8609–8613, IEEE, 2013.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[36] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estima-

tion from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5162–5170, 2015.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[38] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[39] T. Hodaň, J. Matas, and Š. Obdržálek, "On evaluation of 6d object pose estimation," in *European Conference on Computer Vision*, pp. 606–619, Springer, 2016.

[40] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[42] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, "Acquisition of localization confidence for accurate object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–799, 2018.

[43] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[44] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 567–576, 2015.

[45] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[46] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[47] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of operations research*, vol. 14, no. 1, pp. 105–123, 1988.

[48] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *Advances in Neural Information Processing Systems*, pp. 424–432, 2015.

[49] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon,

"Scene coordinate regression forests for camera relocalization in rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2930–2937, 2013.

[50] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 603–619, 2002.

[51] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[52] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 564–575, 2003.

[53] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d object pose estimation using 3d object coordinates," in *European conference on computer vision*, pp. 536–551, Springer, 2014.

[54] M. U. P. Technologies, "Handyscan 3d."

[55] "Vxmodel."

[56] C. B. Duane, "Close-range camera calibration," *Photogramm. Eng*, vol. 37, no. 8, pp. 855–866, 1971.

[57] A. W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, IEEE, 2001.

[58] N. Qian and C.-Y. Lo, "Optimizing camera positions for multi-view 3d reconstruction," in *2015 International Conference on 3D Imaging (IC3D)*, pp. 1–8, IEEE, 2015.

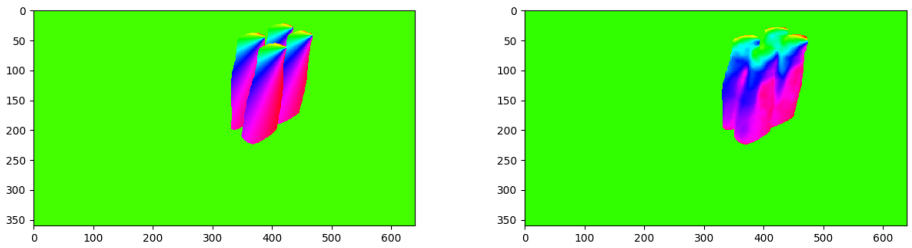
[59] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

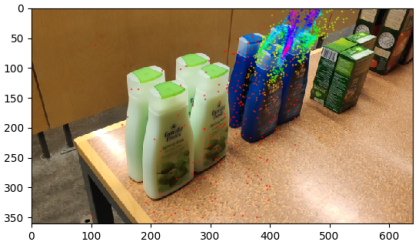
A

Voting field prediction for tightly packed objects

This appendix provide images which underlines the insufficient prediction performance of the neural network, in particular for the voting field in cases of severe self occlusion between objects of the same type.



(a) **(b)**



(c)

Figure A.1: Comparison between the ground truth voting field (a) and the predicted voting field (b) of a keypoint for an image with four tightly placed instances of blue soap. (c) illustrates the image together with the set of hypotheses generated from the predicted voting field (b).

B

Dataset overview

This appendix contains sample images from the dataset, one image for each scene in the dataset.

Validation 1

Validation 2

Validation 3

Training 1

Training 2

Training 3

Training 4

Training 5

Training 6

Training 7

Training 8

Training 9

Training 10

Training 11

Training 12

Training 13

Training 14

Training 15

Training 16

Training 17

Training 18

Training 19

Figure B.1: Overview of all scenes in the dataset. Each figure caption specifies the subset of the dataset (validation or training) and the scene index.

C

Samples of annotation segmentations

This appendix contains samples of generated segmentations in the dataset.



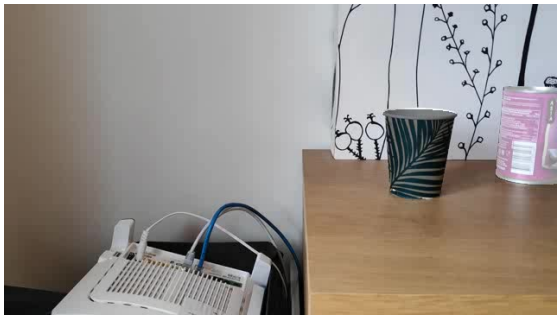



Figure C.1: Overview of produced pose labels. Each figure image is a randomly sampled frame from a randomly sampled video produced by rendering object segmentations onto the original RGB video of a scene, using the poses annotated using our tool.