Universitetet i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Information Technology- Automation and Signal Processing	Spring / ~~Autumn~~ semester, 2019 Open/~~Confidential~~
Author: Harald Thirud Skutvik	………[illegible]……… (signature of author)
Programme coordinator: Professor Karl Skretting Supervisor(s): CTO Torfi Thorhallsson	
Title of master's thesis: Evaluating Computer Vision Methods for Detection and Pose Estimation of Textureless Objects	
Credits: 30 ECTS	
Keywords: Computer Vision, Neural Network, Textureless Objects, Pose Estimation, Object Detection	Number of pages: 66 + supplemental material/other: 1 Stavanger, 11 July 2019 date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Evaluating Computer Vision Methods for Detection and Pose Estimation of Textureless Objects

Master's Thesis
in
Information Technology - Automation and Signal processing
by

Harald Thirud Skutvik

June 2019

Abstract

Robotics, AI and automation; search for these words and two things become apparent. An era of automation is upon us, but even so there are still some simple tasks that grinds it to a halt, e.g. picking and placing objects. These simple tasks require coordination from a robot, and object detection from a computer vision system. That's not to say that robots are incapable of picking up objects, as the simple and organised cases have been solved some time ago. The problems occur in cases where there are no order, in other words chaos. In these cases it is beneficial to detect and find the pose of the object, so that it can be picked up and packed while having full control over the position the object was placed in. This thesis is written at the behest of Pickr.ai, a company looking to automate the picking and packing for retail businesses.

The objective of this thesis is to evaluate available pose estimating methods, and if possible single out one that is best suited for the retail environment. Current state of the art methods that are capable of estimating the pose of objects utilise convolutional neural networks for both detection and estimation. The leading methods can achieve accuracy upwards of the high 90% on pretrained objects. The case with retail is that the volume of available wares may be so large that training on each item is prohibitive. Therefore the testing done has mostly been aimed at the method's generalisability, whether they can detect objects without prior training specific for the object.

A few different methods with varying solutions were examined, from the simpler pure object detectors to two stage 6D pose estimators. Unfortunately none of the methods can be deemed appropriate for the task as it currently stands. The methods do not recognise new objects, and the improvement from limited training does not improve the scores significantly. However, by applying the approaches that are incorporated in the other methods, it may be possible to develop an appropriate new pose estimator capable of handling a retail environment.

Acknowledgements

Some say the thesis marks the end of their masters degree, I feel like its significance is greater than that. It is the culmination of nearly 20 years of education. Nonetheless, with this I finish my 5-year Master's Degree in Automation and Signal Processing at the University of Stavanger.

I would like to thank my external supervisor Torfi and Pickr.ai for aiding me in my work and giving me complete access to everything from knowledge to hardware.

I would also like to give thanks to my internal supervisor Prof. Karl Skretting for his valuable feedback.

Nils Olai E.S., Simen W.T., Gabriel H. and Svein N. for sorting out a couple of spelling mistakes that escaped my attention.

Last but not least, my parents for always being there whenever I have needed them.

Contents

Abbreviations

RGB	**R**ed **G**reen **B**lue,	Common pixel format in pictures
RGB-D	**R**ed **G**reen **B**lue **D**epth	An extension of RGB with depth
ROS	**R**obot **O**perating **S**ystem	Framework for robotic software
JPEG	**J**oint **P**hotographic **G**roup	Picture format with quality loss
PNG	**P**ortable **N**etwork **G**raphic	Lossless picture format
ANN	**A**rtifical **N**eural **N**etwork	Machine learning method
DNN	**D**eep **N**eural **N**etwork	ANN with more hidden layers
CNN	**C**onvolutional **N**eural **N**et	Neural net with convolution
GQ-CNN	**G**rasp **Q**ality **C**onvolutional **N**eural **N**et	CNN focusing on gripping
R-CNN	**R**egion **C**onvolutional **N**eural **N**et	Region focused CNN
RNN	**R**ecurrent **N**eural **N**et	A type of ANN
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform	Feature detection algorithm
SURF	**S**peeded **U**p **R**obust **F**eatures	Improved version of SIFT
GPU	**G**raphical **P**rocessing **U**nit	Component dedicated to graphics
cuDNN	NVIDIA **CU**DA **D**eep **N**eural **N**etwork library	Library for DNN algorithms
SSP	**S**ingle**S**hot**P**ose	Object pose estimator
DOPE	**D**eep **O**bject **P**ose **E**stimation	Object pose estimator
RANSAC	**Ran**dom **Sa**mple **C**onsensus	Parameter estimation method
YOLO	**Y**ou **O**nly **L**ook **O**nce	Object detector
PnP	**P**erspective-**n**-**P**oint	Computer vision technique
SVM	**S**upport **V**ector **M**achine	Machine learning method
COCO	**C**ommon **O**bjects in **Co**ntext	Image dataset
YCB	**Y**ale-**C**MU-**B**erkeley	Dataset by Yale and Berkley
6D	Degreees of freedom	Position and orientation

List of Figures

List of Tables

Chapter 1

Introduction

This chapter serves as an introduction into the content of the thesis. It presents the problem and provides background information, allowing for understanding about how the problem came to be and why it needs to be solved.

1.1 Motivation

1.1.1 A More Automated World

The world is getting increasingly automated, with everything from self driving cars to storage facilities that require little to no aid from workers. The benefits of a robotic workforce are many. They are accurate and flexible, and can be adapted to other uses through for example switching the end effector[1]. Robots are also cost effective and can work day and night. They are actually so cost competitive that industries known for being low cost are looking to expand their robotic workforce [1]. The benefits are many and the possibilities increasing [2]. With this in mind it makes sense to pursue technology and science that can follow this trend, and for companies to look for opportunities to automate. Online shopping companies like Amazon [3] and Komplett [4] are looking to

[1]E.g. switching from a gripper to vacuum suction.

automate as much as possible. As others before them [3, 5], a small start up company by the name of Pickr.ai is looking to do both automated storage and delivery.

1.1.2 Pickr.ai

Pickr.ai [6] is a startup company located on Forus, Norway. They are approaching online shopping and logistics with the intention of cutting costs across the whole value chain. Their current focus is on automating entire warehouses without the need for a complete overhaul and redesign. The intent is to increase profit on online sales such that even low margin products become profitable.

An example of how the process could work would be something akin to this:
An order is received by the warehouse and the system gets to work planning pick order and locates where the different items are. Different robots are activated and move into position in front of the boxes containing the different wares. A camera attached to the robot provides the visuals needed to locate the items. The system uses a method to locate and estimate the pose of the system. From there a pick point is generated that the robot places its vacuum end effector to and picks up the item. Since the system knows the size of the item, it can utilise a packing algorithm to plan where to place it in the order bin. The order bin travels around the warehouse and is filled with wares pertaining to the order. The items are packaged and the order is now ready for delivery, all done automatically. In the future this will be a smaller part of the larger process exemplified in Figure 1.2.

1.2 Problem Definition

The assignment, paraphrased, as defined by Pickr.ai: "Currently a human workforce is needed for the cost and time consuming picking and packing process. Therefore a fully automated system is in development that uses a gantry robot[2], with a single depth camera mounted on the arm to provide visual input. The goal is to evaluate possible methods of locating an item through the use of images of the item within a standardised scene."

[2]Cartesian three axis robot with two additional rotational joints.

More specifically for this thesis is the problem of estimating the pose of a given object given little to no prior knowledge. This was chosen as a starting point since there are no guarantees that sufficient knowledge about the object can be acquired. A suggested solution should surpass earlier methods, and/or add in extended usability. The method(s) put forth should give information about the pose of the object, since further functions rely on this information. The intention of this thesis is not to create a new method, but evaluate methods already available. A picture of the robot can be seen in Figure 1.1. The proposed methods will also be subjected to some constraints that may reduce their usability:

- **Should work on a wide range of objects and be adaptable**: Optimally the proposed method will be able to recognise, or at least handle, a wide range of products. It should therefore be as general as possible, or trained to work in many situations. A solution that only works on a few objects will be regarded as sub par, unless it still surpasses the baseline method. Expanding on that idea, it should be adaptable to more object types if need be.
- **Time, work and upkeep:** Adapting the method to new objects should not require too much time. This is purposefully left vague since it is dependent upon how often the methods would have to be adapted, and how much time Pickr.ai would be willing to invest in upkeep. Following that, the amount of work required to keep the method up to date should be minimal.
- **Data:** Proper amounts of knowledge or data about each object cannot be expected either, because of the time and effort needed to collect this data.

1.3 Problem description

Automation brings with it several problems. In the case of picking wares, be it shampoo, a bag of chips or a can of peas, there's the challenges of locating, pose estimating, picking and robot path planning. For this thesis it is the locating and pose estimation challenges

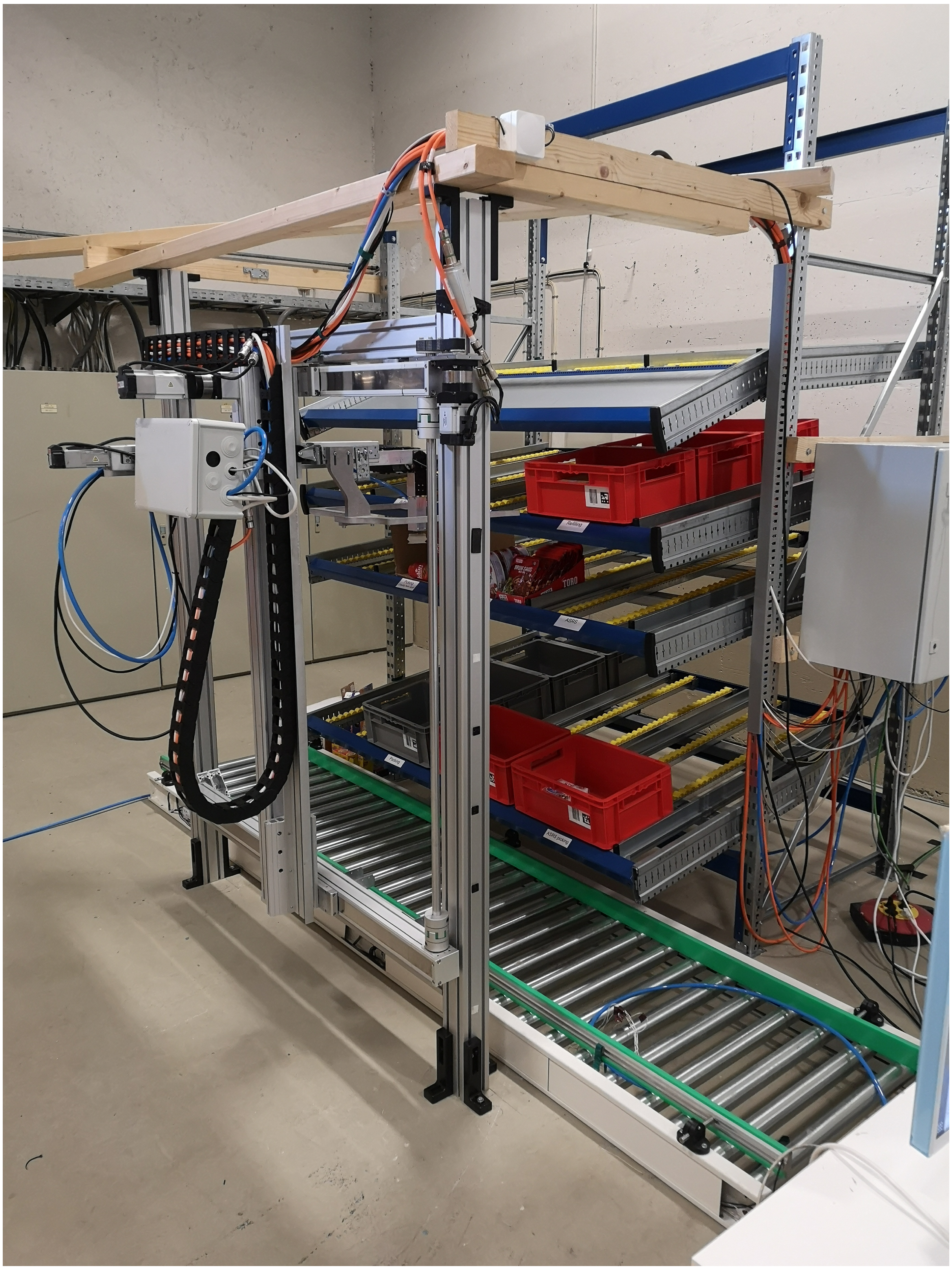


Figure 1.1: The current robot prototype developed by Pickr.ai, here seen in its testing conditions.

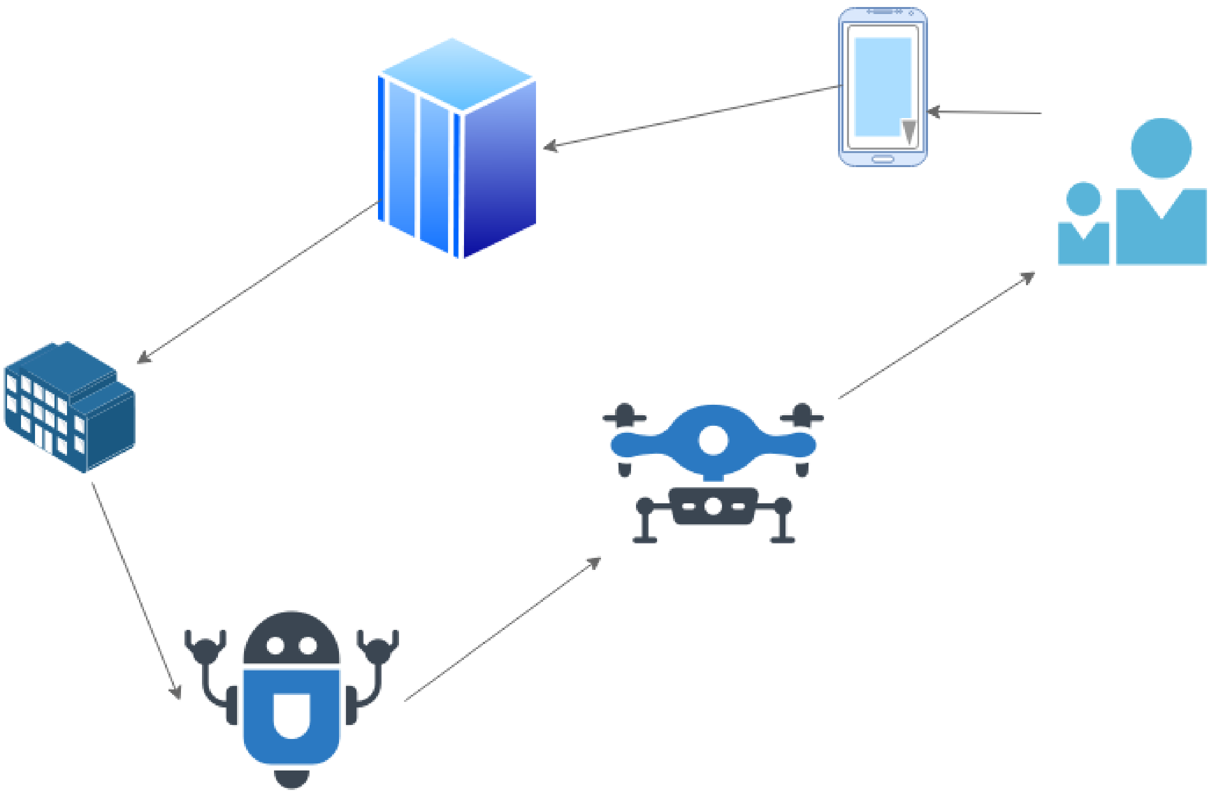


Figure 1.2: A simple overview of how ordering could work

that are the most relevant. When picking, the robot has to be able to find the items, discern one item from another, plan how to move to get to the item and figure out where to grip the item. Using Figure 1.3 as an example. These cans of peas all have the same size, but looking from the top it is difficult to see how large the bottom can actually is. Figure 1.4 shows another potential problem, where it can be difficult to see where one object ends and the other starts. As for textureless objects, a clear bottle of water can be close to invisible. This is something that has to be accounted for if the system is going to be able to handle those clear bottles.

1.3.1 The Bin Picking Problem

In general this problem is called the bin picking problem [7]. There have been made many advancements in regards to robots picking up objects from bins, but the final stop that is random bin picking robots has yet to be reached. What's missing is accuracy. Robots are good at doing repeatable monotonous work, but when the objects are placed randomly

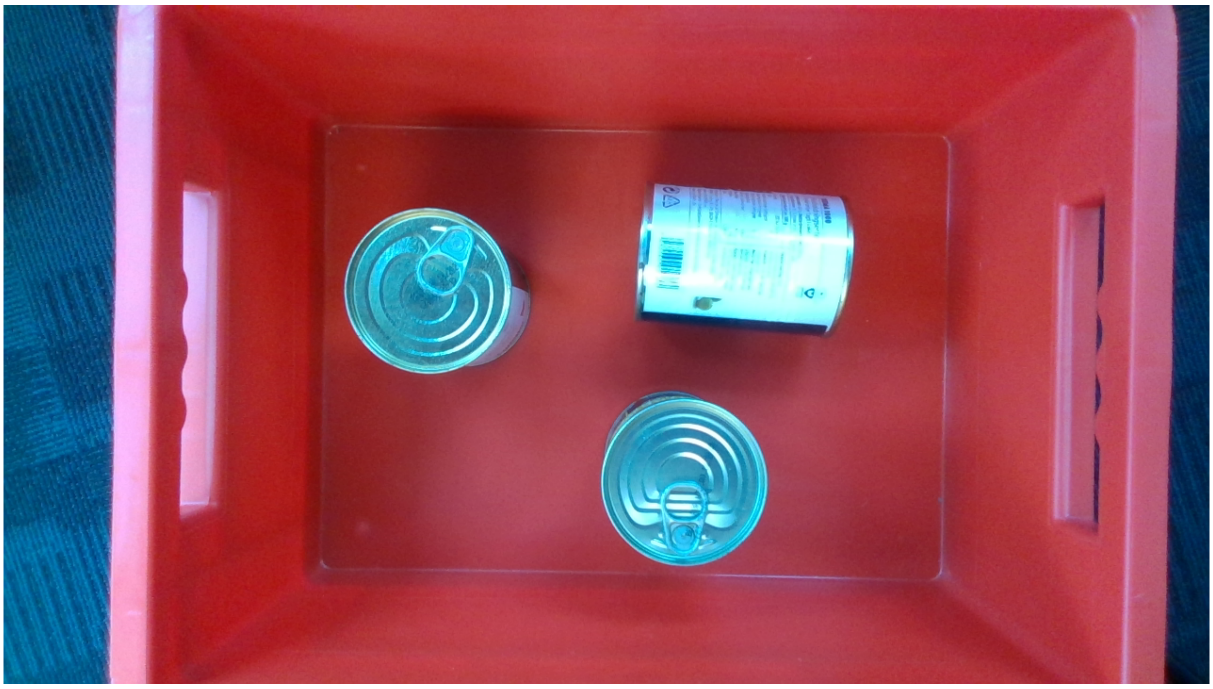


Figure 1.3: Example of what the system may see.

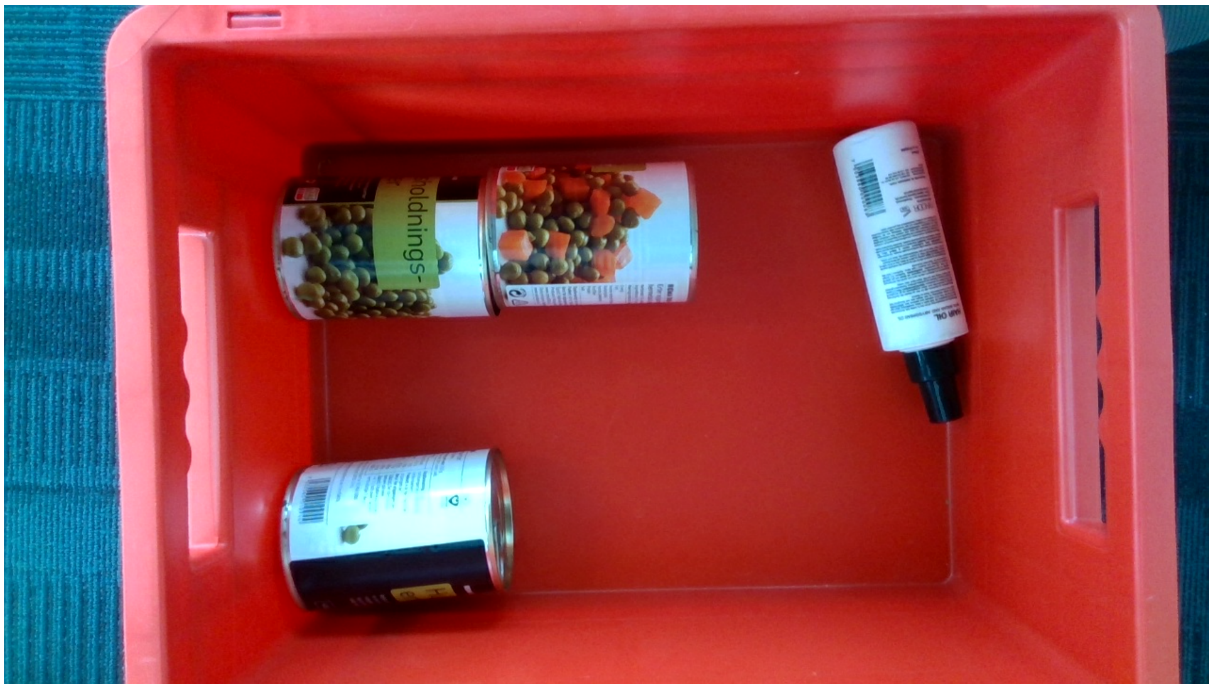


Figure 1.4: Easy for humans, but a computer vision system may have difficulties noticing that there are two cans in the upper left corner and not one long object.

the accuracy comes into play. The goal is to pick up an object in an environment that may change for each pick. To keep up with all of these changes the solution should have a combination of dexterous robots, computer vision, and software and hardware that is powerful enough to compute this in real time.

There are three grades of difficulty of bin picking which is visualised in Figure 1.5:

1) **Structured** - Objects are laid out or stacked in a predictable and organised manner. This makes it easy to register and pick up the objects. Boxes travelling down a conveyor belt with even spacing in a single line is an example of this.

2) **Semi-structured** - The objects are somewhat laid out in a predictable and organised manner. Picking with the aid of imaging is still a relatively easy task. Items dropped on a conveyor belt and thus laying somewhat disorganised is an example of this. The items are still laying in a somewhat predictable matter, on the conveyor belt, but there is a need to register where they are on the belt to be able to pick them up.

3) **Random** - Here the objects are random in every aspect. This means that the orientation, rotation and stacking changes from object to object. This is basically the case when objects have been placed into a bin without regard for anything but them being in a bin. Continuing with the conveyor example, this is at the end when everything is dropped into the bin. The items, be it parts for a car or boxes of candy, are now laying disorganised in a bin. Each time an item is picked up, the top layer might shift and settle down differently. There might also be difficulties discerning one specific item from a mass of items.

Aside from how orderly the objects are placed there is also the problem of the type of objects. A robot, or rather the vision system dealing with the imaging, may have difficulties recognising two boxes simply because they have different texture, much like in Figure 1.4. Same goes for textureless objects, this could be minimalistic designs or see-through objects like plastic bottles. Telling these objects or very thin objects apart may be difficult. As can be seen, or not be seen, where the edge of the product in Figure 1.6 stops and where the background starts. Using a camera with a depth sensor solves the texturelessness to a certain degree, but comes short when applied to low volume objects. The same goes in the case of deformable objects and packaged objects. A bag of chips and a bag of screws are two examples of this. One bag of chips will not be similar to another bag simply because they may have a slight different bend. Similarly a clear bag

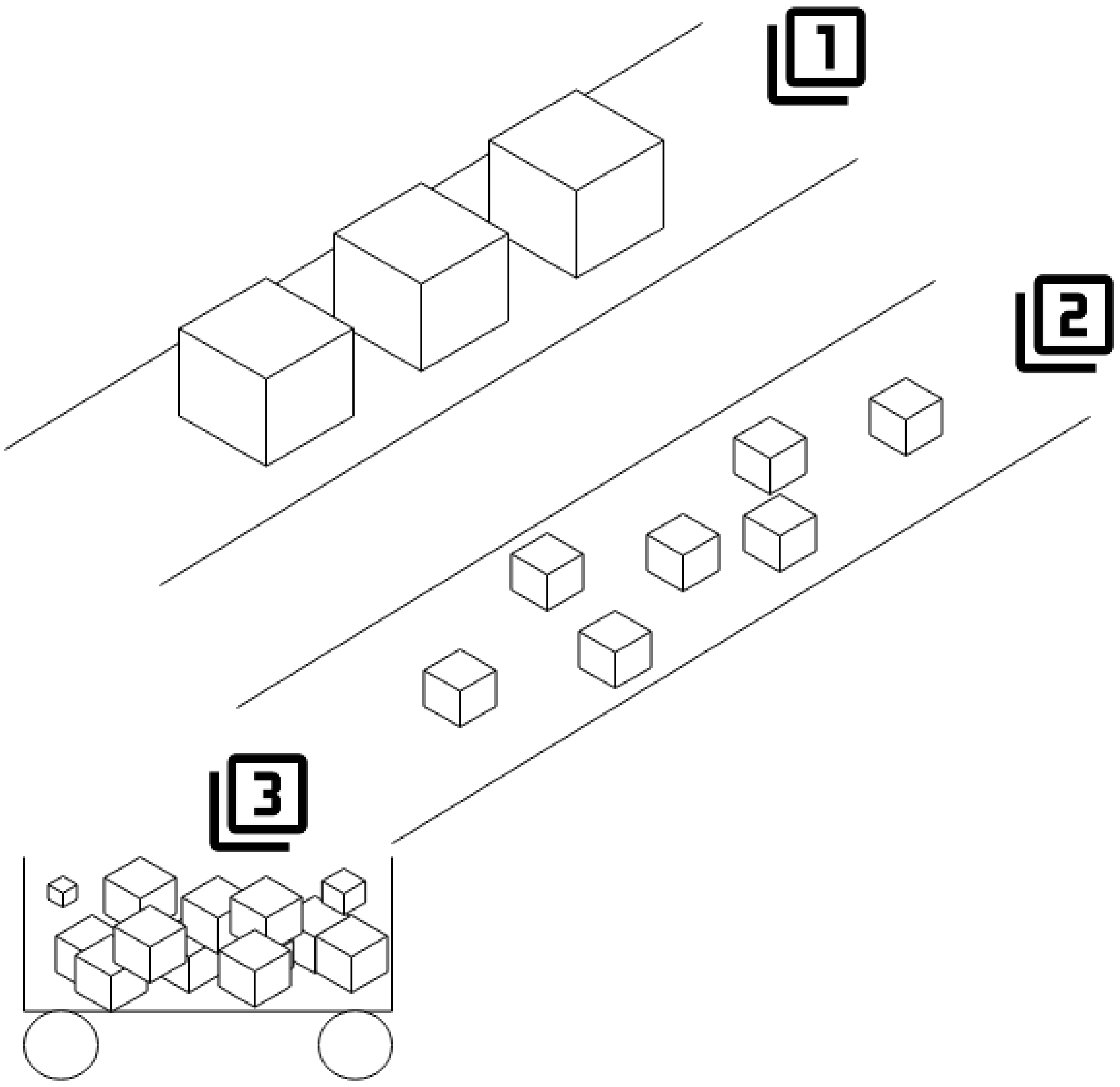


Figure 1.5: Grades of bin picking difficulty

containing screws may prove difficult, since the system may have problems knowing that a screw might be in a bag while others are outside.

1.3.2 Pose estimation techniques

DeepIM [8], Keypoint detector localization [9] and PVNet [10] are examples of the current cutting edge pose estimation methods. All of these utilise some form of neural networks to calculate the pose of an object. The challenge with neural networks is that they require large amounts of data to be sufficiently trained, where the challenge lies in obtaining

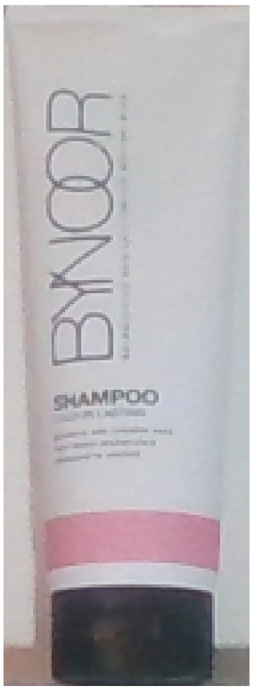


Figure 1.6: One example where the edge blends in with the background to such an extent that is it difficult for a human to notice it.

enough usable data. To recognise several items the net has to be trained on all of those items, and have enough quality data for each of them. Neural networks are trained by giving them examples of what they are looking for, so the more and better examples the better it should perform. Three ways of achieving this is by imaging and labels, data augmentation and synthetic data. Data augmentation artificially expands image datasets by creating modified images, this can be e.g. mirroring, flipping and turning the images. Synthetic data is data that is created and not gathered. An example of this is placing 3D models on an image, Figure 1.7 illustrates how this can be done.

The first data collection method requires access to all of the objects, or datasets containing enough relevant data for a good approximation. Either way a large amount of data in the form of images are required for optimal performance. There's not really such a thing as too much data either, given that it is quality data that is representative of what the network is supposed to learn. The more representative data that is processed, the better a neural network can perform. The amount of data needed also increases fast if the net is supposed to recognise large amounts of products. So, if there are 30 000 different wares in a warehouse, and each item needs over 1000 images to be properly identified[3], then

[3]The pose estimator SingleShotPose uses 1151 images during training to recognise an iron.

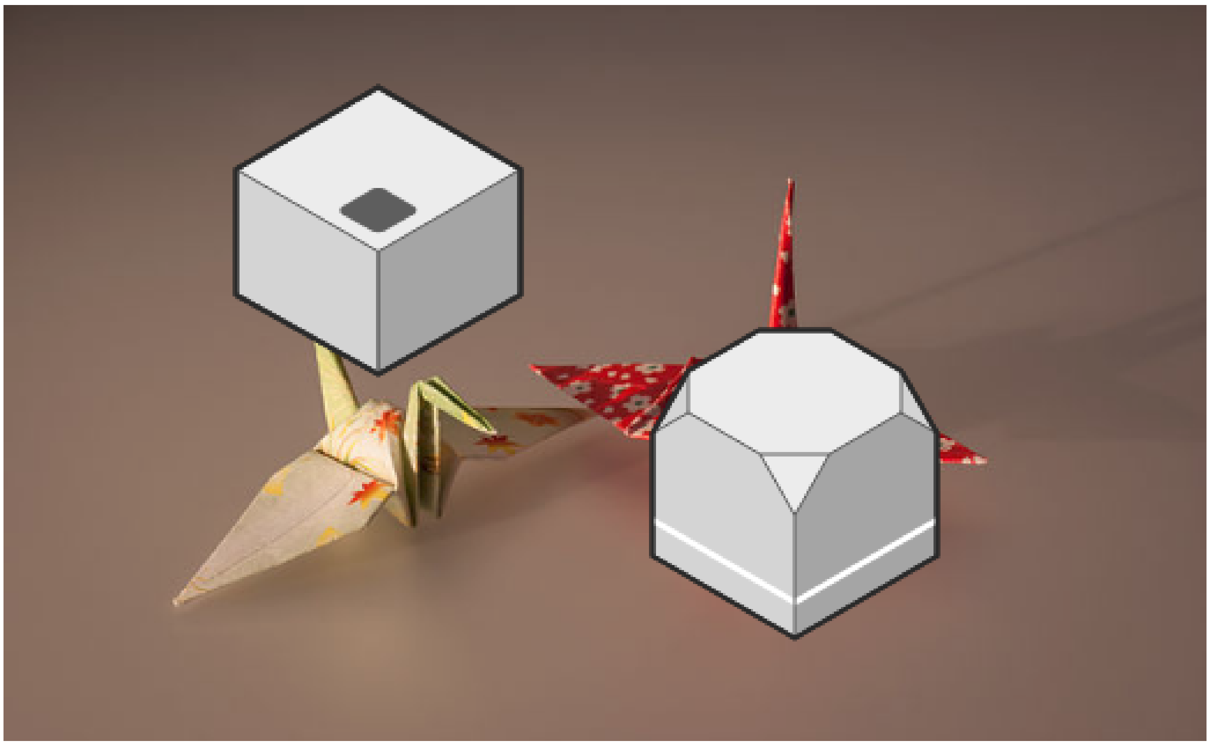


Figure 1.7: Illustration of how synthetic data can be made. Background from Wikimedia commons, public domain

there's going to be at least 30 million pictures. Whether this is possible is a different question entirely, as YOLO9000[11] is trained to detect just over 9000 object classes, but the resulting performance is not necessarily great.

1.4 Outline

The intention of this thesis is to propose a solution that will manage to estimate the pose of objects in a controlled environment, and be a comparatively better solution than the one currently used. Chapter 2 goes into other solutions currently being employed by other companies, and other related work. Chapter 3 expands upon the software and methods that are used and considered, whereas Chapter 4 delves into the testing and observations of the selected methods. Chapter 5 and 6 are discussion and the reached conclusion.

Chapter 2

Related Work

There are two different areas relevant as related work. It is either complete solutions that other suppliers have, that aim to do the same thing as Pickr.ai, or they are methods for localisation and pose estimation. Outright related methods are explored in Chapter 3 as they may be usable solutions for the problem at hand. It is on the basis of that, that most related work mentioned in this chapter is about other complete solutions and not methods.

Pickr.ai isn't the first company looking at automation, be it parts or the whole chain from order to delivery, or are they the only ones looking at bin picking. Dex-Net is an ongoing project to handle object picking, dealing with data generation for datasets and all parts up to optimal grasp quality. There are also several companies trying to automate their warehouses, some noteworthy are Amazon, Komplett.no and Ocado. Amazon are trying to both give incentives to others and develop their own solutions, while Komplett.no bought their solution from Autostore. There are solutions that share ground with what Pickr.ai is looking for, but none that offers everything.

2.1 Company Solutions

The solutions that other companies have developed and are currently in use. None of them deal entirely with the same sets of problems as Pickr.ai, but the areas do overlap.

2.1.1 Autostore

Autostore [12] is a distributor of efficient warehouse solutions that have amongst others provided the storage solutions for Komplett.no. Instead of having rows of shelves, their solution is to make a cube structure that contains all the wares. This allows for a greater storage capacity without the need for more storage space. The grid where all the bins are placed, is advertised as being just a little bit more difficult than LEGO bricks to put together. This means that the transition from a traditional warehouse to one using the solution Autostore provides should be a relatively easy procedure. A shortcoming is that humans are still necessary to deal with the actual picking, as the robots only bring the bins to a picker that then manually picks and packs the order [4]. Nonetheless the setup gives increased speed, efficiency, better overview and control, and a greater storage density [12].

2.1.2 Ocado

In the area of automated warehouses there are a few different companies with different solutions. Most of them employ both human and robotic workers. One of these is Ocado [13], an online only supermarket based in the UK. Their most advanced warehouse, as of May 2018, is located on the outskirts of Andover, England. Their solution is a system where wares are placed in stacks ranging up to 17 boxes high. How these boxes are stacked is algorithmically decided based on the frequency items are ordered. To then reach the items at the bottom requires a temporary restacking of potentially 16 other boxes. The robots are controlled by a central machine, allowing them to work in unison to move the other boxes away when needed.

To facilitate automation of this magnitude in a new warehouse would require a considerable rebuilding, as old solutions with shelves require access from the side, and not the top. This solution is quite space efficient, but is not fully automatic. The machines simply move the crates containing the wares to human workers so that they can pick up the items by hand. The reason for this goes back to a subproblem in bin picking that the robots still have difficulties with, amongst other, bagged items. In the end, the goal of a cheaper and more efficient workforce is achieved.

2.1.3 Universal Robots

Universal Robots [14] is a provider of robotics aimed at more general automation, and not necessarily automation in storage facilities. Of the services their robots may help automate, the most relevant ones are picking and placing, packaging and palletizing. Through the use of their UR+ platform, a degree of automated bin picking is achievable. They say it is simple enough for non-experts to set it up within a few hours. A human operator will, during the setup phase, configure the robots action through a series of wizards. Whether this has to be performed for each new part the robot is meant to pick, is not specified, but the wording suggests that it depends on the objects in question.

Some other provider of pick and place robots are ABB [15], FANUC [16], Yaskawa Motoman [16] and KUKA Robotics [16].

2.2 Previous Research

Previous research and development that to a certain degree reflects the work done either by Pickr.ai, or the solutions looked at in this thesis.

2.2.1 Vision-based Robotic Grasping from Object Localisation, Pose Estimation, Grasp Detection to Motion Planning

This thesis will examine the following recent methods for object localisation and pose estimation:

- YOLOv3 by Redmon et al. [17]
- SingleShotPose by Tekin et al. [18]

- DOPE by Tremblay et al. [19]
- DenseFusion by Wang et al. [20]
- DeepIM by Li et al. [8]

These methods will be described in Chapter 3. For additional details on these and other methods the reader is referred to a very recent (16th May 2019) and excellent review paper by Du et al. [21]. The study goes into detail about the currently available solutions for the whole process, as well as what challenges still remain for the type of approaches the methods propose. Naturally, as the paper looks on the whole process, all of the solutions explored here are also mentioned in review.

2.2.2 Direct Grasp Detection Without Pose Estimation

The Dexterity Network is a research project by Mahler et al. [22] aimed at generating datasets of synthetic point clouds, robot parallel-jaw grasps and metrics of grasp robustness. The project includes code, datasets and algorithms for making these synthetic datasets. The aim of the project is to develop highly reliable robot grasping across a wide variety of rigid objects ranging from industrial parts to everyday items like household items. Each iteration of Dex-Net adds to the previous versions by giving more support and utility [22]. The project does everything but estimate the pose of objects. It can locate and generate a pick up point for a range of end grippers, be it suction or jaw grippers. A more thorough examination can be found in Appendix C.

Chapter 3

Software, Equipment and Methods

This chapter goes into detail about the different methods that were considered, but also the prerequisites needed for the testing including both hardware and software dependencies.

3.1 Software

An overview of the base software that was used, as well as the reasoning behind why one version of the software was chosen above another.

3.1.1 Operating system: Ubuntu

Ubuntu [23] is a free, open source, Linux operating system. It is one of the more popular of the many Linux distributions available. In addition to this there are also different versions available. The two latest long term support ones are 16.04 and 18.04 [23]. In regards to the research, different versions give rise to problems where some software may be made for a certain version, but experimental or not at all available in others.

During the initial research the distribution of choice was Ubuntu 18.04, chosen because it was relatively new and because the initial impression was that the software support was extensive enough. This proved to be a faulty conclusion as the software support turned out to be lacking as some vital software was version locked. Together with user error, this created an opportunity to downgrade to Ubuntu 16.04. This move made it so that the research and development environment was close to identical to what Pickr.ai was using, thus ending in a tested and proven environment[1]. While Ubuntu 16.04 is an older version it is also a long term support version. In this case it means that it will receive standard support until 2021, and security maintenance until 2024 [23].

3.1.2 Programming Language: Python

Python is an object-oriented programming language. It is available under an open source license, making it free to use. There are over 180 000 projects [24] available through `pypi.org`, with some notable mentions as Numpy, Scikit, Matplotlib, TensorFlow and PyTorch. The two latter ones are widely used deep neural network (DNN)/machine learning libraries.

Python comes in several versions with 3.7 being the newest [25]. However, to ensure the most compatibility between all files Python 2.7 was chosen. All methods mentioned later are compatible with 2.7, and in extension the packages they use. That is not to say that 3.x is unsupported, but the authors cannot guarantee that the method will work as they might not have done extensive testing.

3.1.3 Robot Operating System (ROS)

ROS is a type of robotics middleware:

> "... robotic middleware is designed to manage the complexity and heterogeneity of the hardware and applications, promote the integration of new technologies, simplify software design, hide the complexity of low-level communication and the sensor heterogeneity of the sensors, improve software quality, reuse robotic software infrastructure across multiple research efforts, and to reduce production costs." [26]

ROS [27] allows for processes to be run in separate nodes. These nodes may post and receive messages. These messages usually contain the information generated by the node.

[1]Further elaborated in subchapter 3.2.1.

The different messages are posted to topics, so one node may have several messages being distributed through several topics. To get access to the messages, the user has to subscribe to one or more of these topics. This can either be done in a script or one node subscribed to another. For example, one camera node publishes the image stream, and a different node subscribes to the topic containing the image stream. The node may then do object recognition and show the video feed with the object marked. In a larger scope, many nodes may be connected to each other to control a robot, where each joint would be a node. So, instead of creating a complete control system that will only work for that specific setup. ROS allows to "simply" connect nodes and control the system through these nodes. This makes it easier to configure new setups, since all the components are configured to work within ROS already. The ROS ecosystem can be divided into three groups:

- Packages oriented towards applications that use some of the available ROS-libraries.
- Roscpp, rospy and other library implementations of the ROS client.
- The tools used to build, distribute and maintain the ROS-based software.

There are several distributions of ROS, depending on which operating system is used. ROS Kinetic (v10) was made for users on Ubuntu 15.10 and 16.04, while ROS Melodic (v12) was created for the users on Ubuntu 17.04 and 18.04. Cross-platform usage is not supported, but technically possible by building from source [27].

During the initial research and testing phase, ROS Melodic was the version utilised as it was the only version that supported Ubuntu 18.04. Being the newest and recommended version with the longest planned support time gave precedence to choose version 18.04 over the others. However, both due to a change in operating system and software incompatibility, ROS version had to be changed for compatibility reasons. While it is one of the older versions, it still is under support as it is one of the other major releases.

3.1.4 Artificial Neural Networks

An artificial neural network (ANN) [28] is a potentially powerful tool in the category of machine learning. It works by mimicking how a biological neural network works in a general sense. A neural network consists of three general layers, one is the input layer, then some hidden layers and an output layer. Each of these layers consists of one or more nodes that are interconnected, in a way that is reminiscent of how biological neurons connect to each other through synapses. The nodes can transmit signals to one another through these connection, similar to how neurons communicate through the synapses. However the signals in a neural network only move one way from the input layer, through the hidden layers and out to the output layer.

One of the more popular classes of neural networks that is often used in image analysis is convolutional neural networks (CNN) containing several convolutional layers, each consisting of several spatial filters with coefficients learnt from training data [29]. The CNNs belongs in the DNN type of networks that can have several hidden layers, all depending what applications they are used for. The CNN that is shown in Figure 3.1 belongs to the SingleShotPose method. The configuration files reveal that the first convolutional layer is configured to use 32 filters. From the resulting activation map the method alternates between max-pooling and a few convolutional layers to extract the relevant features from the input image. What max-pooling does is downsampling the input by the largest value, see Table 3.1 for an example. The output depends on what the user is looking for. In the case of SSP it'll output 9x2 coordinates representing the centre and the eight corners defining a bounding box around the object. As well as one class value and two range values for X and Y. Similarly with the method You Only Look Once v3 (YOLO), it outputs two corner coordinates x and y, the height and width of the bounding box, a predicted class and the confidences it gives for the class.

The success of a neural net largely depends on how it is built up and the data it is allowed to train on. As most stages of creating a NN, the training also takes some time to complete. This is however dependent on how powerful the hardware is, one which the algorithm is run on. Newer and more powerful hardware is also required for the newer versions of required software.

0	1	2	3
4	5	6	7
8	3	1	4
9	2	5	0

5	7
9	5

Table 3.1: Visual representation of how max-pooling downsampling works, each coloured area is represented by its largest value. Inspiration from `en.wikipedia.org/wiki/Convolutional_neural_network`

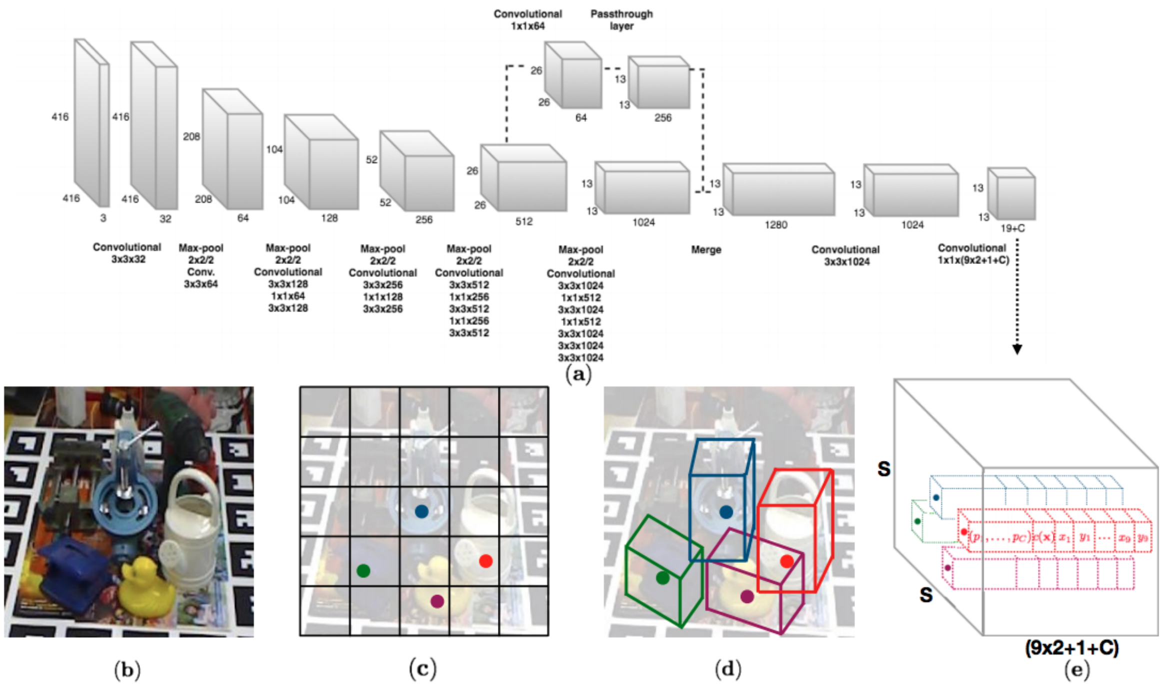

Convolutional 1x1x64
Passthrough layer
416
416
3
416
416
32
208
208
64
104
104
128
52
52
256
26
26
512
26
26
64
13
13
256
13
13
1024
13
13
1280
13
13
1024
13
13
19+C
Convolutional 3x3x32
Max-pool 2x2/2 Conv. 3x3x64
Max-pool 2x2/2 Convolutional 3x3x128 1x1x64 3x3x128
Max-pool 2x2/2 Convolutional 3x3x256 1x1x128 3x3x256
Max-pool 2x2/2 Convolutional 3x3x512 1x1x256 3x3x512 1x1x256 3x3x512
Max-pool 2x2/2 Convolutional 3x3x1024 1x1x512 3x3x1024 1x1x512 3x3x1024 3x3x1024 3x3x1024
Merge
Convolutional 3x3x1024
Convolutional 1x1x(9x2+1+C)
(a)
(b)
(c)
(d)
S
S
(9x2+1+C)
(e)

Figure 3.1: A representation of the CNN used in SingleShotPose, In accordance with MIT Licensing, taken from github.com/Microsoft/singleshotpose [18]

3.1.5 Limiting Factors in Neural Networks

The factors mentioned in the problem definition directly limits much of the machine learning methods [30]. In theory there is no limit to how many objects a neural net can recognise. As long as the net is sufficiently trained it can detect and recognise a large quantity of objects[2]. What limits the number of objects is the amount of data that is available. A neural network will not correctly classify an object without having been trained to find it. It may be able to see and classify it nonetheless, but the classification will in that case not be correct without further processing. Even if there is enough data, the net requires time to train. The more data to train on the more time it will take, depending on how fast the hardware is. However, that there is no need to train it further

[2]For example, YOLO that is mentioned later can recognise up to 80 different classes.

unless a higher accuracy is needed or it has to be expanded with new data. Whenever the material that the network calculates on changes or is added to, it will have to be retrained to work on the new data. The aforementioned upkeep may be time consuming, be it gathering and preprocessing new data or retraining. Adding to the preprocessing, it may take longer if labelling has to be done manually as well.

3.1.6 Training

Training is the process of adjusting the weights of the network to obtain a desired classification result. In supervised learning, each training sample is presented together with a class label, often obtained by manual labelling. In general, the more data a neural network can be trained on, the better are the chances of it performing well. Neural networks runs a risk of overfitting if the data is insufficient essentially learning the training examples with poor generalisation to novel same-class examples. A model is overfitted to a dataset when small changes to said dataset results in a large variance. Ideally this variance should be as low as possible, so that changing a data point results in only a minor shift. Bias can also affect the results, when trying to approximate complex patterns to a too simple model, which happens when the data does not lend itself to be simplified because of its complexity [31]. Not necessarily applicable to DNNs, but something to keep in mind for simpler machine learning algorithms. Combating overfitting may be done by dividing the data into three parts of train, cross-validation and test set, and as mentioned increasing the amount of representative data [32].

Generally data can be anything, and any given neural net is designed to handle a specific set of inputs. In the case of methods discussed later in the chapter, e.g., some can only handle images in the JPG file format and labels in PNG file format. A solution to not having enough representative data is to train the model on a larger but still relevant dataset, for then to train the model additionally on a smaller dataset. This is called transfer learning [33] and borrows from the idea that humans are capable of using previously learned knowledge and apply it to new tasks. By transferring knowledge learned in one task, to a related task and thus improving learning of the new task.

3.1.7 cuDNN

NVIDIA CUDA Deep Neural Network library (cuDNN) [34] is a library of primitives that allows for GPU-accelerated computations for deep neural networks. cuDNN provides implementations for standard routines use by DNN such as forward/backward convolution, pooling, normalisation, and activation layers. The framework is used by many researchers and is thus necessary when testing the more advanced methods in this thesis. It is advertised to GPU-accelerate several deep learning frameworks, such as TensorFlow and PyTorch, which are utilised by the methods that will be examined later. The newer versions of CUDA, which is the toolkit, require better hardware. For comparison, an old NVIDIA GeForce 660M GPU can only run version 3.0, while the servers at the University of Stavanger have NVIDIA Tesla P100s, allowing for CUDA version 10.0. Not only will the newer versions improve computational time, but it is also required for certain functions that the older versions lack.

A different platform that is available is OpenCL[3]. OpenCL is an open source platform implemented by amongst others AMD. However, since none of the tested methods implements OpenCL, it is only mentioned here as an alternative.

3.2 Equipment

Some information about the robot and the camera line that was used.

3.2.1 Intel RealSense D400 series

The Intel RealSense D400 series [35] is a popular line of low-cost depth cameras. The D400 series uses stereo vision to calculate depth, but also features an RGB sensor for colour images and an infrared projector to aid the depth perception. See Appendix A for technical specifications about D415 and D435. The nodes realsense_camera and

[3]www.khronos.org/opencl/

realsense_camera2 publishes several different topics, that in general falls into parameters, depth image stream, colour image stream and infrared.[4]

The RealSense cameras have a ROS package that allows for easy integration into ROS projects. The realsense package only supports Ubuntu 14.04 and 16.04, making it the main reason for the switch from Ubuntu 18.04 to 16.04, as the workarounds necessary to make it compatible simply shifted the problems further down the line. The package can be can be found at `github.com/IntelRealSense/librealsense`.

3.2.2 The Robot and the Storage Solution

The current iteration of the robot is a cartesian, three axis, gantry robot with two additional joints. It is to be mounted in front of shelves as seen in Figure 1.1. The storage racks that it is designed to be attached to are regular frames, with a slight decline allowing for the crates to roll towards the robot. This allows for the currently used crates to be replaced on the backside of the shelves.

3.3 Considered and Current Solutions

This section goes into detail about the different computer vision based methods that were considered, as well as the baseline methods used for comparison.

3.3.1 Baseline: Cylindrical Pose Estimation

The current, and therefore the baseline, method used is an adaptation of Random Sample Consensus (RANSAC) [36, 37]. It is an iterative method that is used to estimate the parameters of a mathematical model in observed data without being affected by outliers. This means that the method is capable of ignoring some degree of noise. In regards to pose estimation it is used to fit a cylindrical shape over the object in question. The pose estimation uses the PointCloud data and overlays the cylindrical shape. The noise in this

[4]Taken from topics listed when running the D415.

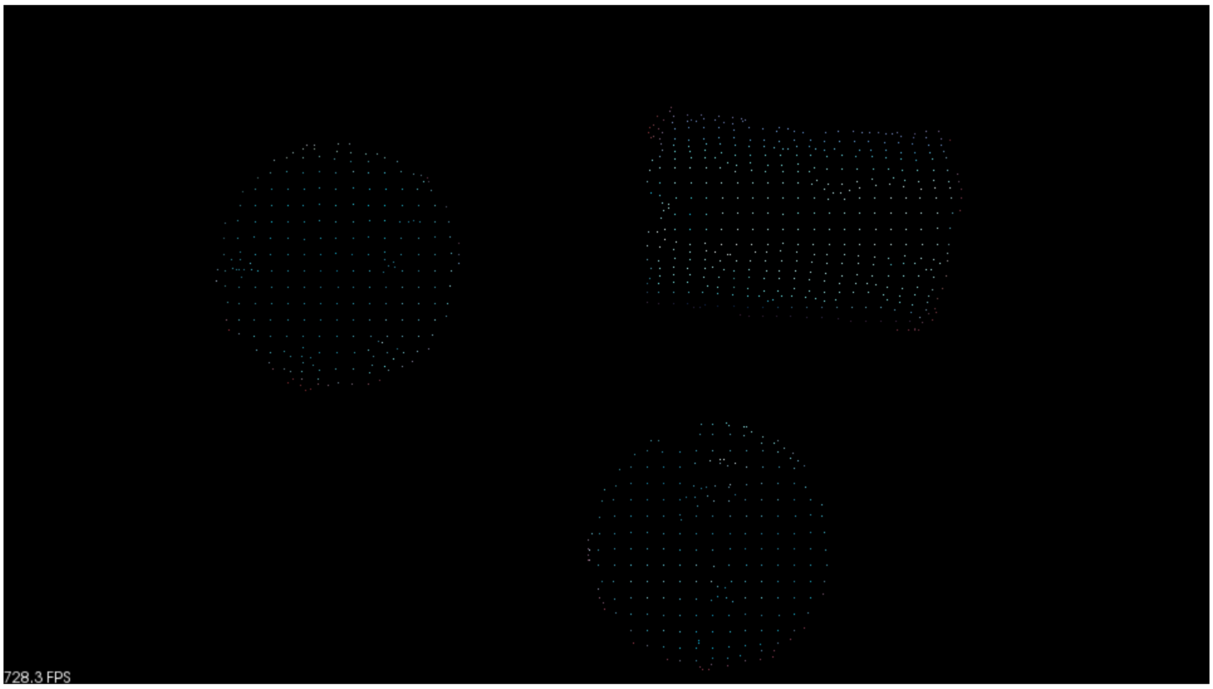


Figure 3.2: The PointCloud data where RANSAC can be applied, here the edges of the box are removed as well. The PointCloud here is a representation of the image in fig.1.3

case is therefore points that fall outside of norm, e.g. if a point on Figure 3.2 is much larger than the others surrounding it.

This technique is both intuitive and easy to visualise, but the simplest might not be the best solution. The estimator in its current form will also only work on cylindrical objects. That is not to say that it would not work on other shapes, something it would be able to do, and has been done with circular shapes. Another caveat however, is that it can only check for one shape at the time, so the estimator will only be able to find items that fits its current pattern. That is unless the database contains knowledge about the shape of the object, and thus allows for the system to use the correct shape for the estimation. In a less than optimal situation, the system might have to cycle trough several shapes before finding the correct one.

3.3.2 Baseline: SIFT & SURF

One of the previously considered solutions are the Scale-invariant feature transform (SIFT) [38] and Speeded up robust features (SURF) [39] methods, and the later improvements.

SIFT is a feature detection algorithm that detects and describes local features in an image. SIFT can be used for object recognition, video tracking, match moving, 3D modelling and more. These properties makes SIFT a tempting choice for both detecting objects and pose estimation especially since the algorithm is invariant to both scale and orientation. The caveat, however, is that it requires features to work. Features meaning details in this regard, something that textureless objects by definition lacks in any meaningful amounts. Furthermore, SIFT does not provide a pose estimation, but this is something that could be implemented separately. SURF is regarded to be a more robust and faster algorithm to SIFT. It covers similar tasks as object detection, image registration and 3D registration. The two algorithms are based on the same principles as well, but details in steps are different. These differences do however not make up enough of a difference to work well on textureless object.

Both of the algorithms would also need one or more sample images of the products they are supposed to detect. The reason is that both algorithms need a reference to compare to. The references also need to cover the whole object, meaning that there are pictures from several angles. Which is similar to neural networks, but for other reasons. Neural networks train to give a probability of a certain object being in the image, while SIFT/SURF compares the points in the image to references they already have to tell if the object is in the image.

3.4 Neural Network Methods

The two previously mentioned classical methods provide a baseline for the more recent neural network methods considered in this thesis. The development of neural networks for computer vision is a fast moving field. Finding the state of the art methods is possible by looking at the web page paperswithcode.com. It has an assortment of papers about several tasks within computer vision problems, and one of these is 6D pose estimation. The page uses leaderboards to measure methods up against each other, and topping these charts are methods using neural networks.

3.4.1 Datasets

LINEMOD[5] and Yale-CMU-Berkeley (YCB)[6] are two datasets that are used both for training and to measure accuracy of methods. Some others are the Occlusion dataset, which is a relabelling of the LINEMOD dataset and the Imagenet dataset that YOLO is pretrained on. The datasets used for object detection are generally significant amounts of pictures[7] taken of objects with accompanying labels, masks and bounding boxes. These large dataset allows for much more training without running the risk of overfitting the networks, as explained in the training section. In addition to having images of the objects the more advanced methods also require a 3D model and bounding box.

The accuracy of the 6D pose estimation methods are generally measured using an average distance score called ADD, Equation 3.1, which was first used by Hinterstoisser et al. [40],

$$ADD = \frac{1}{m} \sum_{x \epsilon M} \left\| (Rx + t) - (\hat{R}x + \hat{t}) \right\| \tag{3.1}$$

where m is the number of points on the 3D model and M is the set consisting of all 3D points of the model. $[R|t]$ is the ground truth and $[\hat{R}|\hat{t}]$ is the estimated pose. Through the use of this metric it is possible to compare the different methods against each other when applied on the same dataset. It is this metric that will allow for comparison between the different methods, and is also one of the most often used metrics in pose estimation [41].

A different accuracy measure is the mean average precision. The score goes from 0-1 and conveys how well the predicted boundary box fits with the ground truth boundary box. When the overlap goes over a certain threshold it is regarded as a match. The average score for different thresholds is what makes up the mean average precision score. It is one of the primary scores used to compare object detectors but not the one primarily used here. Therefore, unless otherwise stated, the meaning behind accuracy refers to the ADD score.

[5]http://campar.in.tum.de/Main/StefanHinterstoisser
[6]http://www.ycbbenchmarks.com/
[7]Imagenet has in total $\approx$ 14.2 million pictures, image-net.org.

3.4.2 Single Shot- and Two-Stage Detectors

Object detection using neural networks come in three main variants, although one could be said to just be a subgroup of the other. Two-stage detection, Single Shot Detection [42] and YOLO are the variants of deep learning-based object detection. Regions with CNN features(R-CNN) [43] and its variants are examples of two stage detectors because they divide the process into one feature extraction part, and one predicting part. In the case of the original R-CNN the features are found by first using a selective search algorithm[8], the regions found are then sent into a CNN. The CNN outputs a feature vector which is fed into a Support Vector Machine (SVM). The SVM then classifies the presence of an object within the proposed regions. Calculating all of this takes time, both for training and object detection. The Fast R-CNN presented in [43, 44] improved training time by 9x and test time with 213x. The Fast R-CNN uses the image as a whole as input into the CNN, and not just regions.

R-CNNs can be very accurate but they are slow compared to single shot detectors, obtaining 5 frames per second compared to the 45 to 155 frames per second that a single shot detector can achieve [45]. This speed does come at a price though. Looking at YOLO, a family of single shot detectors, the YOLO9000 version could only achieve a 16% mean Average Precision. Delivering quantity over quality, since it could classify 9000 different object categories [11]. Since then improvements have been made with YOLOv3 [17]. R-CNNs are not the only two-stage detectors developed, and while YOLOv3-tiny achieves speeds up to 220 frames per second [17], some two-stage methods are achieving a good balance with a real time computation of 30 frames per second. High frame rate isn't strictly necessary for the system, so accuracy is therefore a more relevant measure to look at. Figure 3.3 shows the current accuracy rankings on the LINEMOD dataset populated by both both two-stage and single shot methods.

See Appendix C for a more software based examination of the following methods.

[8]Combines several candidate regions into larger ones.

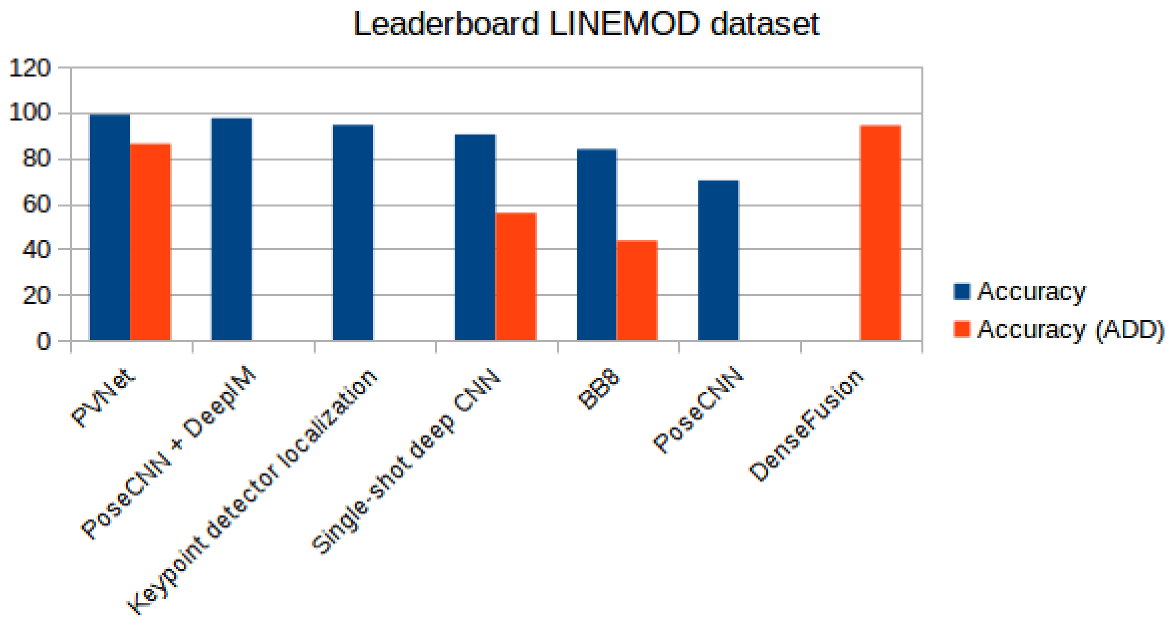


Figure 3.3: The current rankings of LINEMOD on the paperswithcode.com leaderboards

3.4.3 YOLOv3

The newest iteration of YOLO called YOLOv3 by Redmon et al. [17] is a type of single shot detector. It is one of the simpler neural net methods, and doesn't require CUDA or similar software to be tested. YOLOv3 can quickly be implemented in Python with OpenCV, and is quick enough to work on a live image stream, accomplished on an ageing processor[9]. YOLOv3 predicts bounding boxes through the use of dimension clusters. Together with the bounding boxes it supplies a class and its probability [17].

What YOLOv3 does not include is an estimation of the pose of the object, as is simply an object detector as can be seen in Figure 3.4. Therefore YOLOv3 cannot be used on its own, but it may still prove a useful starting point.

3.4.4 SingleShotPose

Building on the more simplistic YOLO object detector, there is the more advanced SingleShotPose (SSP) by Tekin et al. [18] SSP is a single shot detector that detects

[9]Seven year old Intel Core i7-3630QM.

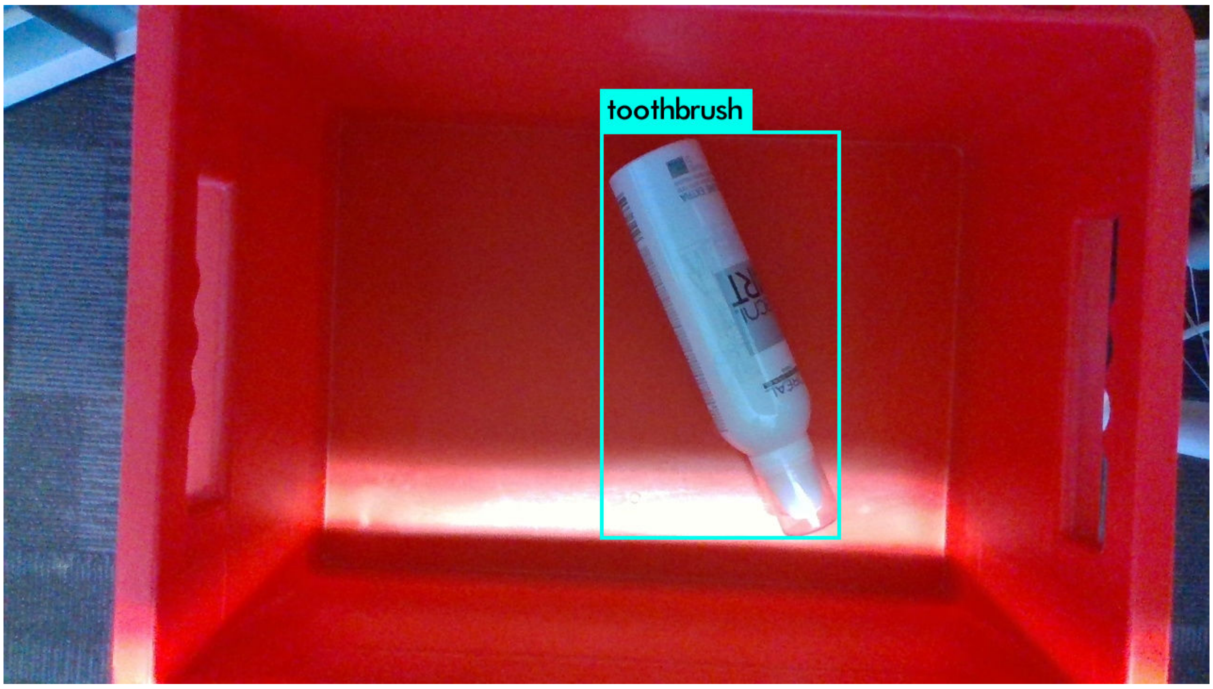


Figure 3.4: A simple test using YOLO to detect an object, however it did miss on the actual type of object.

objects and tries to predict their 6D pose. Since it takes advantage of the single shot detection principle, there is no need for multiple stages or hypotheses. The method uses a CNN architecture that is inspired by YOLOv2, it directly predicts the 2D image locations of the projected vertices of the objects 3D bounding box. Then the object is 6D pose is estimated through the use of a Perspective-n-Point (PnP) [10] algorithm [18]. At the time of writing, SSP is the highest ranking method on the OCCLUSION dataset, on the leaderboards found at [41]. SSP is also among the top four methods on the LINEMOD dataset leaderboard.

Usage is not as straight forward as with YOLOv3 as it is not possible to pass an image stream to SSP. SSP can probably be modified to accept an image stream, but as it stands now the testing has to be done by creating a new dataset and testing through the included evaluating functionality. One example of this test is shown in Listing 3.1.

[10]See https://en.wikipedia.org/wiki/Perspective-n-Point for an overview.

```
2019-06-12 15:17:45    Testing glue...
2019-06-12 15:17:45    Number of test samples: 1036
-----------------------------------
  tensor to cuda : 0.000487
         predict : 0.005314
get_region_boxes : 0.008109
            eval : 0.009506
           total : 0.023416
-----------------------------------
2019-06-12 15:18:35 Results of glue
2019-06-12 15:18:35    Acc using 5 px 2D Projection = 96.62%
2019-06-12 15:18:35    Acc using 10% threshold - 0.0176 vx 3D Transformation = 44.79%
2019-06-12 15:18:35    Acc using 5 cm 5 degree metric = 55.41%
2019-06-12 15:18:35    Mean 2D pixel error is 2.441506, Mean vertex error is 0.033075, mean corner erro
2019-06-12 15:18:35    Translation error: 0.032786 m, angle error: 5.442948 degree, pixel error:  2.441
```

Listing 3.1: The results running SSP on a trained object

3.4.5 DOPE

A solution with native ROS support has the advantage of easy integration and adaptation to a robot system. One such method is Deep Object Pose Estimation (DOPE) [19] made by Tremblay et al. DOPE is a ROS package for detection and 6D pose estimation. It only works for known objects[11] with input from an RGB camera. The pretrained objects are all from the YCB dataset. Figure 3.5 is an example of DOPE in action.

The fact that DOPE already has native ROS support makes it a strong contender for further testing. It works well on the six objects it has been trained on, but as it is a specific method it will require training to recognise new objects. The way DOPE is trained is through the use of synthetic data [19]. Originally this data comes from the YCB dataset, and from there generated images resulting in a dataset known as Falling Things (FAT) [46]. Pickr.ai looked into this and found the time necessary to train far exceeding the time frame they envisioned.

[11]Cracker box, sugar box, tomato soup can, mustard bottle, potted meat can, and a gelatin box.

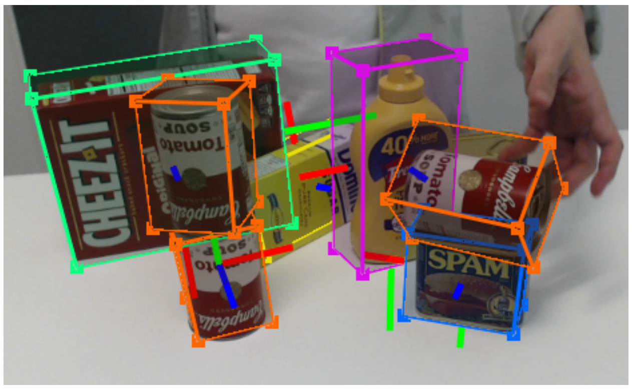


Figure 3.5: Example of DOPE in use, used in accordance to Creative commons licensing [19]

3.4.6 DenseFusion

An RGB-D camera can, as mentioned before, register both colours and depth. So far the methods mentioned are only using the colour image for the estimation. Therefore, in order to make use of as much of the available data as possible, the method DenseFusion was researched. Wang et al. [20] made DenseFusion which is a 6D object pose estimator that takes an RGB-D image as input and tries to predict the pose of the objects in the image. This method fully leverages both RGB and depth when estimating the pose of a known object. Just as the other neural network methods, DenseFusion uses a CNN for estimation. But what differentiates it from other methods is the fact that it combines the two data sources, RGB and depth. After the initial per-pixel prediction, the method goes through an iterative process to improve the predictions. This improvement, seen in Figure 4.1, raises the average accuracy from 86.2% to 94.3% on the LINEMOD dataset [20]. At the time of writing, DenseFusion is currently the leading method on the YCB-Video dataset [41].

3.4.7 DeepIM

What none of the previous methods have claimed to be capable of, is to give an estimation of an unseen object. DeepIM [8] stands out in this regard, as Li et al. [8] claims it is able to estimate the pose of unseen objects from the ModelNet[12] dataset. So while [11, 18, 20]

[12] `modelnet.cs.princeton.edu`

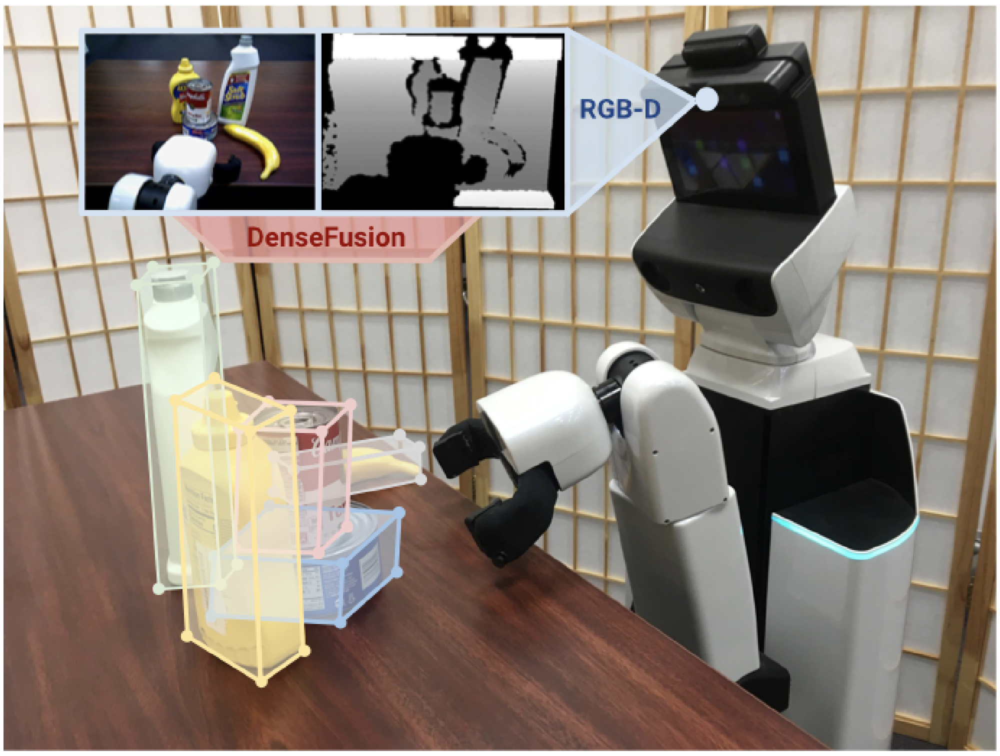


Figure 3.6: Visual explanation of how DenseFusion works 1/2. In accordance to MIT licensing [20, 20]

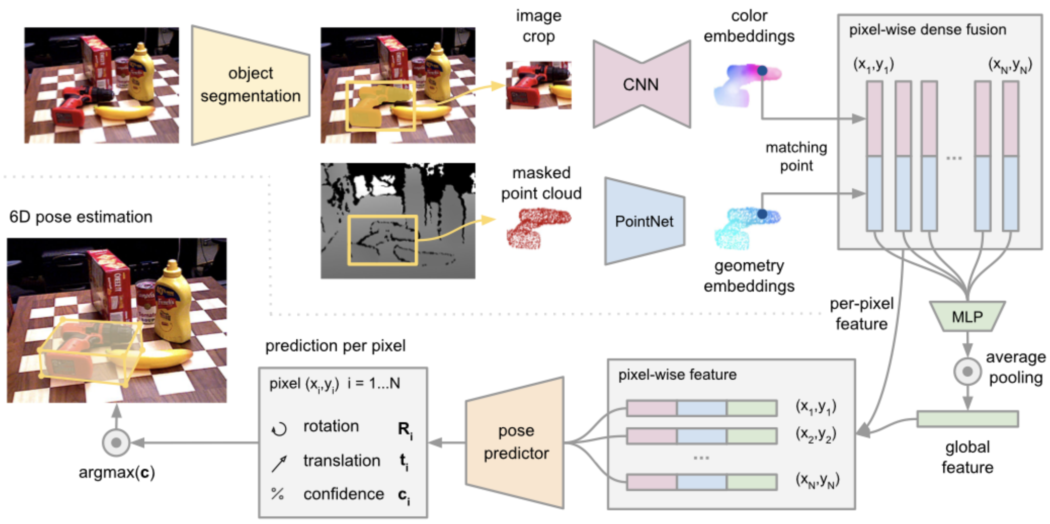


Figure 3.7: Visual explanation of how DenseFusion works 2/2. In accordance to MIT licensing [20, 20]

all are capable of some form of object detection and estimation, only DeepIM suggest that it is capable of handling unseen objects. As for known datasets, DeepIM is at the time of writing ranked number two [41] on the LINEMOD dataset when used in cooperation with a method called PoseCNN [47]. The reason PoseCNN is utilised is because DeepIM doesn't actually handle the initial pose estimation, it simply refines the pose found by a different pose estimator. Therefore, to use DeepIM a primary pose estimator is required.

3.4.8 Dex-Net

Even though Dex-Net was mentioned in related work, there are some merit to further investigate if it can be applied to the problem at hand. So while it doesn't actually predict the 6D pose of the object, it is quite capable of grabbing an item it hasn't seen before. As stated in Appendix C about Dex-Net 4.0, it achieves a pick success rate of 95% [22, 48]. Therefore Dex-Net may prove to be useful paired with a pose estimator, as they don't necessarily provide a pick point by default. Dex-Net also allows for robotic path planning and it can also be acquired as a ROS package.

Chapter 4

Experiments and Evaluation

This chapter delves into the experimental part of the thesis. It goes into how the experiments were conducted and the reasoning behind the execution of the experiments. The results of the experiments will also be presented, as well as the results from the original authors to provide context. Seeing the test results in light of the original results allows for evaluation of the methods.

4.1 Experimental Setup and Dataset

This section goes into the way the experiments were supposed to be, and was conducted. As well as how the preprocessing was done.

4.1.1 Initial Setup

The initial setup consisted of the RealSense D415 depth camera placed above a red box that was to be filled with different wares, see Figure 1.3. This resembles how the visual system operates on the robot, but lacks the moving parts as they aren't an integral part to the object detection and pose estimation. Although the robot isn't an integral part of the object detection, the system it operates on is arguably worth implementing. As mentioned in Chapter 3 about ROS, it works by having nodes send information through the use of topics and messages. Although this adds to the complexity of the testing, it can be argued that it is necessary. How well a given method works may be influenced by this complexity, and gives an indication of the performance in a real life scenario.

One semi plausible example of how the system can grind to a halt, is when there are several ongoing CPU intensive operations. One node publishing a PointCloud2 message, that then is translated into an image message by a different node, has caused a system freeze. Faster hardware can of course alleviate this problem, but it is nonetheless worth keeping in mind. With this premise in mind any method would have to be able to handle a continuous image stream or snapshots from the stream, and to then process this information and pass it along to other nodes. Optimally without causing lag in the system.

This idea was however hampered by the ways the different methods work. This is because the methods were not necessarily created with robotics in mind, or at least ROS. While DenseFusion, Dex-Net and DOPE all have varying levels of ROS integration, the other methods like YOLOv3 and SSP has none. This means that a ROS node, or something similar would have to be made to integrate the other methods into an ROS environment. A requirement for this is that the methods only require an image and/or depth data[1] to work. But this is not the case either, SSP and DenseFusion cannot be evaluated without also having access to the ground truth/labels for each image sent to the method. That is the situation with the released code, as the authors at least in the case of DenseFusion has a unreleased method that does not need the ground truth. For the released method it would essentially mean applying the bounding boxes of the items in real time, to then use the methods on them. This would of course not work since the sole purpose is to find these bounding boxes[2] in close to real time.

4.1.2 Revised Test Setup

One thing most of these methods have in common, besides pose estimation, is the datasets they have been trained on, and in extension what they are evaluated up against. Some

[1]Like Point Clouds

[2]More correct would be the 6D pose, but the bounding box can be regarded as an extension of this and is also in part what the methods find.

widely used dataset in regards to pose estimation are Common Objects in Context (COCO)[3], LINEMOD[4] and YCB[5]. For object detection there is ImageNet[6] which YOLOv3 is pretrained with. If two or more methods are tested up against the same dataset, then it is possible to compare them to each other.

In extension to this, the evaluation scripts that give these results and metrics can then be tested the same way with new data. As long as the data is properly formatted. E.g. the way SSP does it with 21 values, where all the coordinates are normalised and the first number pointing to which class it belongs to. Testing this way gives an indication as to how well the method would work in an actual application, but does not say anything about how easy it is to customise it and integrate it into existing systems. Not all methods could be tested on new data though, as some have no official way or guidelines of how to create a new dataset.

The goal of the tests is firstly to see if the methods will work without any specific training towards the new objects that the methods will recognise the objects as something that the method is already trained on. The methods are only trained on a few items, therefore it is not feasible for them to classify the objects correctly. A correct classification is however not necessary as long as it can give an estimation that is within reasonable limits, e.g. estimating a tube with something similar like a bottle. With similar shapes and sizes, the resulting bounding boxes should have enough resemblance that they can be used interchangeably. By post-processing the results it could be possible to get an accurate pose, but that hinges on the methods producing viable results.

Stage two is improvement, where the methods are trained on the test data through the use of transfer learning. The hypothesis is that the minimal training will give some improvement over the non-training results. Proper training would guarantee a higher level of accuracy, as can be seen in Table 4.1. But to reach that level of accuracy, more images and time are needed, and this violates the constrictions that the methods must adhere to as mentioned in Chapter 1.

[3]cocodataset.org
[4]campar.in.tum.de/Main/StefanHinterstoisser
[5]ycbbenchmarks.com
[6]image-net.org

4.1.3 Test Data

Even though the test setup changed, the test data is still acquired from the original test setup. However, it is now done by making a dataset with 103 pictures, Figure 4.1a shows an example from the dataset. These pictures are then used to create a ground truth with masks and bounding boxes, Figure 4.1c. In this case it was done manually to make sure the results were correct, but masks could for example be generated through automatic segmentation. SSP only requires images and corresponding labels for evaluation, and the masks are used during training.

The methods only accept labels in a specific format. To account for this in the case of SSP, a script was used that extracted the four corner coordinates of the bounding box, and then generated a plausible 3D bounding box. The centre point is also generated from the original bounding box.

In the case of DenseFusion the tool called LabelFusion was utilised. To label with LabelFusion it is necessary to have a connected depth camera, and a 3D model of the item that is to be labelled

Labelling

Preprocessing is a time consuming process and since there are restrictions on time usage, it needs to be surveyed. The dataset made for this project has 103 pictures, where 88 contain single items and 15 contain more than one item. Each item in the pictures has to be given a bounding box and a mask. Total labelling time was 2.4 hours with a mean of 82 seconds per picture. This is an upper estimate, as the program did not account for non-effective work. This means that small breaks counted towards total time. These labels are then reformatted via a script into the correct format.

4.1.4 Test Dataset

The test dataset was provided by Pickr.Ai and contains about 103 pictures, of which some representatives can be seen in Figure 4.2. The dataset consists of bottles of approximately

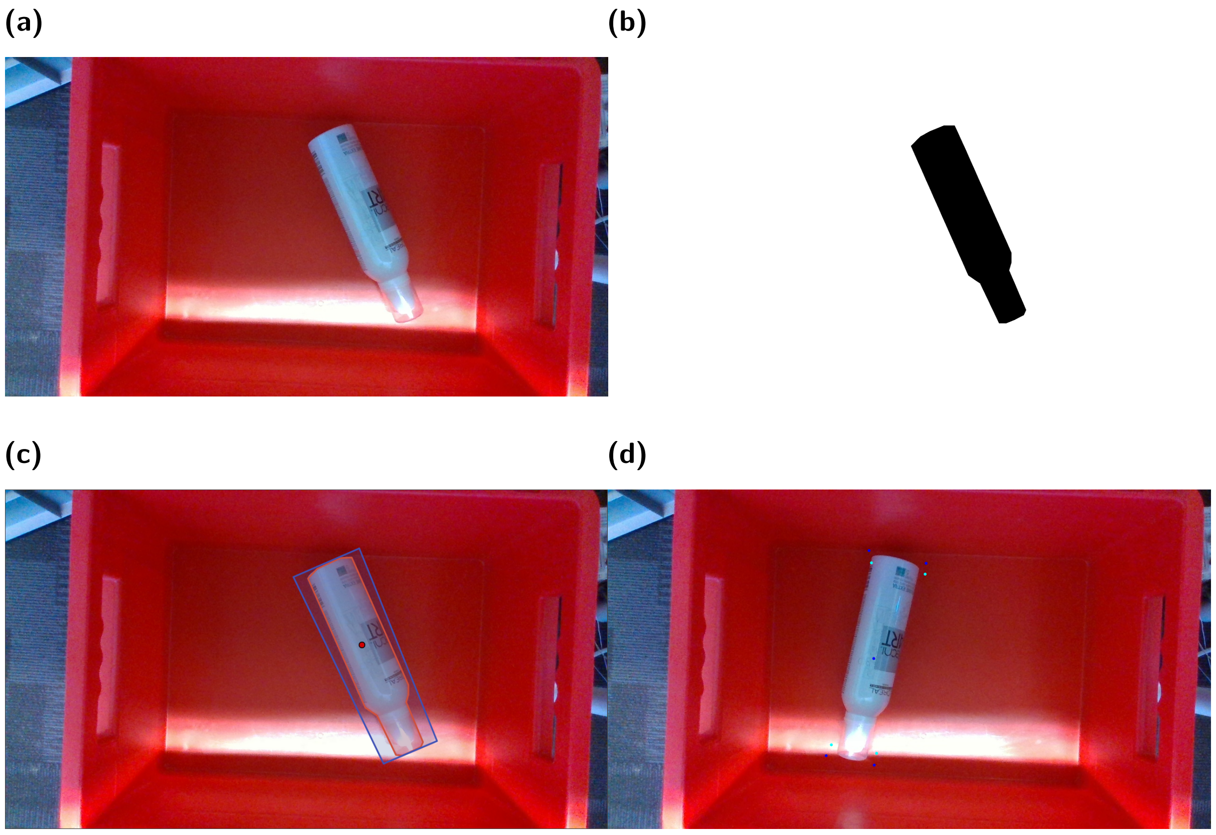


Figure 4.1: a) One of the pictures in the test dataset
b) Example of a mask used during training
c) Mask and bounding box layer overlaid on top of Figure 4.1a
d) Test picture with the original bounding box in blue and the generated in cyan, as well as the centre point generated from the original box

the same size and shape, however the colouration differs between the different brands. There are also nine images that contain more than one product. These are not included in the training, just as a precaution as SSP only trains with one correct item at the time.

4.2 Experimental Results

A summary of the results gathered from the in-house experimenting, as well as the results from the official testing. The official results have also been verified before testing of the test dataset.

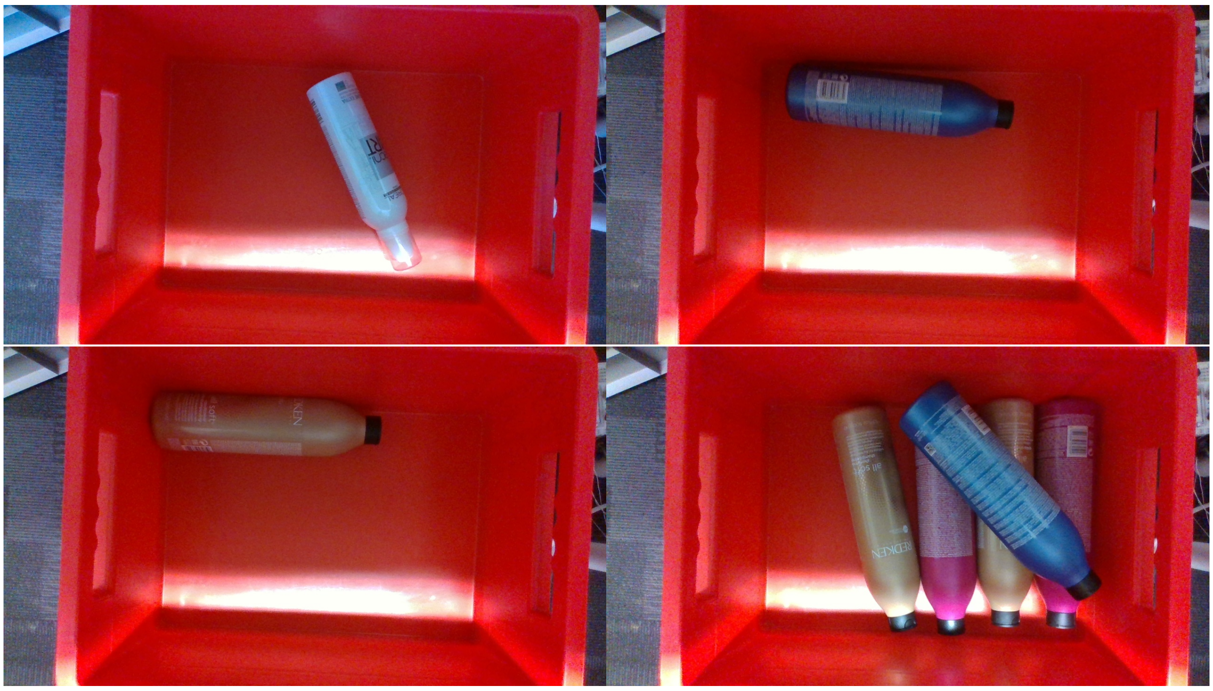


Figure 4.2: Some representative images from the test set

4.2.1 LINEMOD Dataset

Methods that did not allow for training on a custom made test data set, at the time of writing, were instead tested on the LINEMOD set This was the case with DeepIM and PoseCNN for reasons discussed in Appendix C. Some results from the methods can still be acquired through the use of e.g. LINEMOD. These results show how each method fares against the others given the same input, and at the same time confirms the results mentioned in the papers. As the numbers in Table 4.1 have been matched with tests done in-house, so that a comparison can be made.

4.2.2 SSP: Test Dataset

Table 4.2 shows the results when running SSP without training it on the test dataset, as well as the results of SSP running testing on the Glue object from the LINEMOD dataset. Looking at the results from both the unseen object and the properly trained object, it is easy to see some large differences. The results show that SSP is not estimating the correct pose of the objects. The errors on the test dataset are also magnitudes above what the

Method	**RGB**		**RGB-D**	
Object	SingleShotPose	DeepIM+PoseCNN	DenseFusion (per-pixel)	DenseFusion (iterative)
Ape	92.10	98.4	79.5	92.3
Benchwise	95.06	97.0	84.2	93.2
Cam	93.24	98.9	76.5	94.4
Can	97.44	99.7	86.6	93.1
Cat	97.41	98.7	88.8	96.5
Driller	79.41	96.1	77.7	87.0
Duck	94.65	98.5	76.3	92.3
Eggbox	90.33	96.2	99.9	99.8
Glue	96.53	98.9	99.4	100.0
Holepuncher	92.86	96.3	79.0	92.1
Iron	82.94	97.2	92.1	97.0
Lamp	76.87	94.2	92.3	95.3
Phone	86.07	97.7	88.0	92.8
Average	90.37 ($SD = 6.90$)	97.5 ($SD = 1.5$)	86.2 ($SD = 8.2$)	94.3 ($SD = 3.5$)

Table 4.1: Comparison of accuracy(ADD) between SSP, DeepIM and DenseFusion on the LINEMOD dataset [8, 18, 20], all numbers are percentages.

properly trained results are. Furthermore, the difference between untrained and trained is mostly small but noticeable, with a trend towards improvement after training.

Acc 5 px 2D projection: Pose estimate is regarded as correct when the mean distance between the 2D projections and the ground truth is less than 5 pixels[18].

Acc 10% threshold: A pose estimation is taken to be correct if the mean distance between the true coordinates of 3D mesh vertices and those estimated given the pose is less than 10% of the objects diameter[18].

Acc 5 cm 5 degree metric: Much like the ADD metric, but instead of having the accuracy based upon the size and shape of the object, it has here been set to 5cm and 5°[49].

No official account for the other metrics were found, but they seem like they are key number taken from the other metrics.

	Untrained	Retrained	LINEMOD glue
Acc 5 px 2D Projection	0	0	96.62
Acc 10% threshold	0	0	44.79
Acc 5 cm 5 degree metric	0	0	55.41
Mean 2D pixel error	11476	5332	2
Mean vertex error	0.615	0.471	0.033
Mean corner error	272.86	267.17	3.57
Translation error	0.611	0.467	0.033
Angle error	108.72	107.18	5.44
Pixel error	11476	5332	2

Table 4.2: The results from testing SSP on the test dataset, and glue from the LINEMOD dataset

4.2.3 DenseFusion

The test dataset was supposed to be tested on DenseFusion as well, so to be able to compare it to SSP the same way as official testing is done. However, formatting the test data proved to be a difficult task as there were no clear way of formatting it. Therefore a temporary test set was made using LabelFusion, which allowed for about 300 labelled pictures to be gathered from about 15 seconds of film. Although this is one of the recommended labelling tools[7] [20], it did not produce data that could be used directly for training. The preprocessing in the form of formatting the label data was not sufficient for exporting it into DenseFusion. The results from the trial run is only useful as proof of concept, as it failed to find anything. This is most likely to be due to imprecision in the data labels, as some guesswork had to be done because of a lack of information regarding what the different metrics represent.

An attempt was also made to use the YCB dataset and organising the test dataset similarly. It wasn't the full dataset, but only a small part of it that still ended up with about 265 GB of data. The test run on the YCB dataset was however not finished. The testing has gone on for about a week of continued training, and has yet to finish[8].

[7]See `https://github.com/j96w/DenseFusion` for mention
[8]As of the time of writing this, June 9th.

Chapter 5

Discussion and Future Work

This chapter discussed the suitability of the considered methods in light of the test results reported in the prvious chapter. It also explores some ideas surrounding neural networks and how they can be developed to better estimate the pose of unseen objects.

5.1 Choice of Methods

The reasoning for primarily choosing SSP and DenseFusion for testing is as follows. SSP is a 6D pose estimating method that leads the Occlusion leaderboard, and ranks high in the LINEMOD rankings. DeepIM is the current leading method of the LINEMOD leaderboard. It could be of interest for further testing, since results give the impression it can handle unseen objects with some success. The figures presented by DeepIM suggest that it mostly finds the outline of the objects [8]. Looking back at how the robot operates, it needs a pick point for which to pick up the object. SSP finds a centre point of the object it is estimating [18]. This centre point may be possible to use as a pick point, after some post processing. While not a defining reason, it did contribute to the selection of SSP.

DenseFusion is the other main method taken into consideration. It leads the YCB-Video leaderboard, albeit with just a 0.1% lead above PoseCNN combined with a iterative post refinement method. It is also one of the newest methods available having been released 15. Jan 2019. It was chosen because it leverages both RGB and Depth data for pose estimation [20]. When having access to two sources of input data, it is reasonable make use of both and not ignore one.

Dex-Net was proposed at an early stage of this research and would have been a good candidate had it not been for the fact that is does not actually estimate the pose of an object. Dex-Net 4.0, as mentioned in Chapter 2, has a high success rate of more than 95% on 25 novel objects [48]. By default it also proposes a pick up point depending on the gripper used. Therefore, Dex-Net in parallel with a different method could give good results.

YOLOv3 is primarily an object detector, so it does not provide an estimated pose. Nonetheless, SSP is built on a YOLOv2 implementation [18], so with this in mind it seemed worth investigating if YOLOv3 could detect objects from the test dataset. If it did, then that would give cause to modify it to work as a pose estimator.

SIFT and SURF were mostly considered because they do not require training and they are already implemented in libraries like OpenCV [50]. As touched upon in Chapter 3, neither SIFT nor SURF work well with textureless objects. It is for this reason that SIFT and SURF are only mentioned. Worth noting as well is that the methods provide only keypoints, that will need to be matched to a model to infer the 3D pose.

5.2 Future work

Seeing as the work done just gives an overview of potential solutions, there is much work that can be done by going in depth into one or all presented methods. The section also includes a concept that might fit the overall goal that Pick.ai was looking for.

5.2.1 Continued Evaluation

The methods tested in this thesis does not represent an exhaustive list of all suitable methods. Furthermore, as the technology progresses and matures, there is bound to be more pose estimators created. Therefore, there could be several methods in the coming years that are good candidates for further evaluation. Fully implementing or rewriting the methods evaluated in this thesis could also be put up for consideration. However, in order to test several methods the thorough testing had to be skipped, as the process would have been too time consuming.

There is also the case of other non neural network methods. While SIFT & SURF may not be suitable due to texturelessness, expanding upon the RANSAC method could work for other shapes than the cylindrical. Other computer vision approaches could also be worth investigating. Just because neural networks are on the rise, does not mean that simpler solutions should be disregarded as they can be used together with a more advanced catch-all solution.

5.2.2 Concept: General 6D Pose Estimation Method

As will be further expanded upon in the conclusion, there are no methods capable of achieving a proper 6D pose estimation of unseen objects. However, most of the methods bring with them functionality and features which may be of use for a general pose estimator. The following concept should therefore include:

- Use of RGB-D data
- Transfer learning
- Use of synthetic data
- Post processing
- (Pick point)

According to Li et al. [8], methods that use both RGB and Depth outperform pure RGB methods unless some form of post refinement is applied. Therefore, by using DenseFusion as either the framework or a template can be recommended, as it allows for the complete RGB-D data to be used. Using an existing method is not necessary, but may save time since it is already proven to work.

First and foremost in regards to training, is the use of transfer learning. All previous methods, apart from Dex-Net, utilise some form of transfer learning. Transfer learning is used heavily as it allows for training on a large and representative dataset, and is therefore recommended. A synthetic dataset may so be used to further specialise the method. Tremblay et al. [19] mention that synthetic data helps with proofing the method against extreme lighting environments. Even if the environment the robot operates in is controlled, making it able to handle a wider selection of environments can be argued to be mostly positive. A downside with synthetic data is that is has to be generated, which takes time, and will only be as as good as the original objects it is based on.

The main problem with current pose estimators is that they, for the most part, can only handle objects they have been trained on. Solving this can theoretically be done by training on general shapes instead of specific items. By categorising items into general shapes, like spheres, cylinders, boxes etc., the method may be capable of providing a pose estimation that fits the object approximately. Meaning that e.g. both a bottle of soda and a bottle of shampoo is estimated to be equally large cylinders. The consequence of this is that some detail is lost, but theoretically it should not impact the packing algorithm nor the actual packing procedure. This is because the detail has already been reduced into the height, width and depth of the bounding box. Training on geometry and not RGB is however an unsolved challenge, but one that should be manageable since depth data already is used in DenseFusion [20].

Both DeepIM and Dex-Net may be beneficial to implement. Using DeepIM in post processing to refine the pose may counteract the less detailed pose the concept above produces. While not a necessity, the improved pose would allow for a more precise packing of the items. Implementing Dex-Net could be considered, as it is capable of handling both the robotic path calculation and the generation of pick points. Dex-Net has already proven itself with good results [22], so an implementation could act as a second layer of robustness. In the event of a false negative where no object or pose is procured, the implementation of Dex-Net could be used as an backup to at least pick and pack an item[1].

[1]This may again cascade into further problems, as the packing algorithm now have an unknown object in the packing crate.

Chapter 6

Conclusion

The goal of the thesis was to evaluate existing pose estimating methods, and to see if any of them are suitable for locating an item within a standardised scene, as well as estimating the pose of said item. The method in question should be general and adaptable, and should work fast so to be efficient. It should not require much work to maintain, and should work with small amounts of available data.

From the results, or rather the lack of results, and other observations done through the work on this thesis, it cannot be concluded that any method tested is viable in their current state. They either cannot estimate the pose of the item, as they only work on pretrained items, or they would require a considerable amount of time to train, so that they are trained for each specific item. Even if the time was found to properly train one of the methods, it would require constant retraining as designs are subject to change. If this work keeping the neural network up to date is feasible given the necessity for both time and hardware, is up to Pickr.ai.

The testing that could be accomplished on the test dataset gave mostly poor results, further underlining that the neural networks tested are not suitable. The fact that the methods proved difficult to test and train is also worth taking into account. That does not however conclude that neural network based methods are ill advised for this problem, just that a different route may be the better choice[1]. What has not been taken into account

[1] E.g. a method like the one conceptualised in Chapter 5

is how much work it would take to implement one of these methods. Observations do however indicate that it may be substantial, which is the same reasoning for the change in how the experiments were conducted.

Appendix A

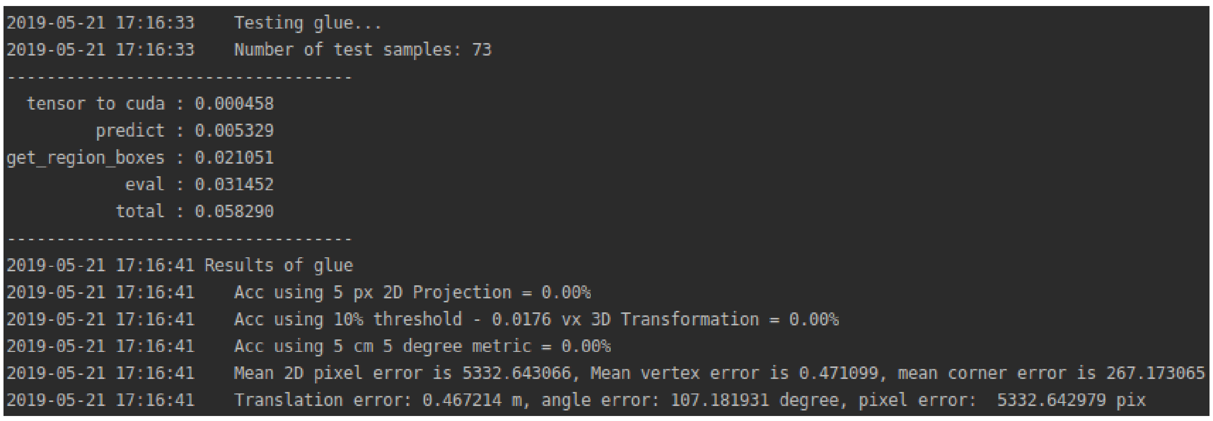
Intel RealSense D400 series

	Intel RealSense Depth Camera D415
Environment	Indoor and outdoor
Depth Technology	Active infrared (IR) stereo
Image Sensor Technology	Rolling shutter: 1.4 μm x 1.4 μm pixel size
Main Intel RealSense™Products	Intel®RealSense™vision processor D4 Intel®RealSense™module D410
Field of View (FOV)- (Horizontal x Vertical) for HD 16:9	63.4°x 40.4°(+/- 3°)
Depth Stream Output Resolution	Up to 1280 x 720
Depth Stream Output Frame Rate	Up to 90 fps
Minimum Depth Distance (Min-Z)	0.16 m
Maximum Range	Approximately 10 meters
RGB Sensor Resolution & Frame Rate	1920 x 1080 at 30 fps
RGB Sensor FOV (Horizontal x Vertical)	69.4°x 42.5°(+/- 3°)
Camera Dimension (Length x Depth x Height)	99 mm x 20 mm x 23 mm
Connector	USB Type-C
Mounting Mechanism	One 1/4-20 UNC thread mounting point Two M3 thread mounting points

	Intel RealSense Depth Camera D435
Environment	Indoor and outdoor
Depth Technology	Active IR stereo
Image Sensor Technology	Global shutter: 3 μm x 3 μm pixel size
Main Intel®RealSense™Products	Intel®RealSense™vision processor D4 Intel®RealSense™module D430
Depth Field of View (FOV) (Horizontal x Vertical) for HD 16:9	85.2°x 58°(+/- 3°)
Depth Stream Output Resolution	Up to 1280 x 720
Depth Stream Output Frame Rate	Up to 90 fps
Minimum Depth Distance (Min-Z)	0.11 m
Maximum Range	Approximately 10 meters Accuracy varies depending on calibration, scene, and lighting conditions
RGB Sensor Resolution & Frame Rate	1920 x 1080 at 30 fps
RGB Sensor FOV (Horizontal x Vertical)	69.4°x 42.5°(+/- 3°)
Camera Dimension (Length x Depth x Height)	90 mm x 25 mm x 25 mm
Connector	USB Type-C*
Mounting Mechanism	One 1/4-20 UNC thread mounting point Two M3 thread mounting points

Appendix B

Results



```
2019-05-21 17:16:33    Testing glue...
2019-05-21 17:16:33    Number of test samples: 73
----------------------------------
  tensor to cuda : 0.000458
         predict : 0.005329
get_region_boxes : 0.021051
            eval : 0.031452
           total : 0.058290
----------------------------------
2019-05-21 17:16:41 Results of glue
2019-05-21 17:16:41    Acc using 5 px 2D Projection = 0.00%
2019-05-21 17:16:41    Acc using 10% threshold - 0.0176 vx 3D Transformation = 0.00%
2019-05-21 17:16:41    Acc using 5 cm 5 degree metric = 0.00%
2019-05-21 17:16:41    Mean 2D pixel error is 5332.643066, Mean vertex error is 0.471099, mean corner error is 267.173065
2019-05-21 17:16:41    Translation error: 0.467214 m, angle error: 107.181931 degree, pixel error:  5332.642979 pix
```

Figure B.1: The best result from testing on the test dataset using SSP.

```
Testing glue...
Number of test samples: 73
-----------------------------------
  tensor to cuda : 0.000407
         predict : 0.004782
get_region_boxes : 0.014281
            eval : 0.007301
           total : 0.026771
-----------------------------------
Results of glue
Acc using 5 px 2D Projection = 0.00%
Acc using 10% threshold - 0.0176 vx 3D Transformation = 0.00%
Acc using 5 cm 5 degree metric = 0.00%
Mean 2D pixel error is 11476.583008,
Mean vertex error is 0.615241, mean corner error is 272.864685
Translation error: 0.611315 m, angle error: 108.721245 degree,
pixel error:  11476.583136 pix
```

Listing B.1: The results running SSP without training on the test dataset

```
Testing glue...
Acc using 5 px 2D Projection = 0.00%
Acc using 10 px 2D Projection = 0.00%
Acc using 15 px 2D Projection = 0.00%
Acc using 20 px 2D Projection = 0.00%
Acc using 25 px 2D Projection = 0.00%
Acc using 30 px 2D Projection = 0.00%
Acc using 35 px 2D Projection = 0.00%
Acc using 40 px 2D Projection = 2.74%
Acc using 45 px 2D Projection = 4.11%
Acc using 50 px 2D Projection = 4.11%
```

Listing B.2: Testing with the multi-object weights

Appendix C

Software

C.1 YOLOv3

Software

YOLOv3 can be run on any system as long as it has Python , OpenCV and Numpy. It is possible to either run it by command line or implement it in a Python script. The only extras required is the YOLOv3 model and optionally the example image.

Usage

The command line testing can be done after downloading the files from `https://pjreddie.com/darknet/yolo/`, a tutorial is also available there. When it comes to using YOLOv3 in a script, a good tutorial can be found at `https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/`.

Training was not tested as YOLOv3 does not provide the 6D pose of item, but a tutorial explaining how to train is also on the official YOLO page. However, just running YOLOv3 on an image stream from e.g. a webcamera shows that it is both fast and powerful for what it is made for. It was capable of finding several chairs and the people sitting on it, as well as computers and cups all while working near real time on an aging computer.

C.2 SingleShotPose

Software

The research group tested the code on Linux running CUDA v8, cuDNN v5.1, PyTorch 0.3.1 with the code being Python 2.7[1]. The testing on UiS's servers was with CUDA v10.0.130, cuDNN v7.5, PyTorch 0.3.1 and Python 2.7. Under the MIT License the software is free to use without limitations as long as the licensing agreements are met.

Installation is straightforward, mostly requiring downloading training data and pretrained weights that sums up to about 9.6 GB. The Python Libraries numpy, scipy, PIL and opencv-python are also required to run the code. As for the code itself it is attained through git cloning of the repository found at `github.com/Microsoft/singleshotpose`.

Usage

Training and testing the software can be done right after the environment is set up [2]. Since the software contains already trained weights[3], there is no need to train before testing. Retraining the weights is simply done trough terminal commands. Optionally there is also pretraining of the model, this also through one command.

It is worth noting that there is one set of weights for each object and one set of weights for multiple objects. This means that a dataset can either be tested on one by one set of weights, or for all at the same time. Training works the same way whether it is single or multi-object weights.

The way this code is made does not lend itself to be implemented into other projects without major rewriting. The software works as a contained project, meaning that it can train and test without much extra work. However, to expand upon the usability and adapt it to new challenges may be difficult. An example of the standard output from SSP can be seen in Listing 3.1. To be able to show these results, there is a need for labels as well. This limits testing and usage to datasets consisting of images that have labels.

[1]There is also one version using Python 3.0 which is PyTorch 0.4 compatible
[2]All prerequisites installed, software and datasets downloaded
[3]Pretrained on the LINEMOD dataset

Labels

The labels are in .txt format consisting of 21 values. The first value denotes the class of the object. The second and third are the x and y coordinates for the centre point of the object. Following are eight coordinate (xy) pairs marking the corners of a 3D bounding box. The last two are the x- and y-range, which are the max length and width of the object given xy coordinates. All values but the class value are normalised by the image dimensions, meaning x/image_width and y/image_height. Some code to format and prepare these labels have been attached to the thesis.

C.3 DOPE

Software

The code has been tested on Ubuntu 16.04 with ROS Kinetic. Unless the system has a full ROS installation, there might be some ROS packages that needs to be downloaded. As for Python libraries, DOPE requires pyrr, PyTorch, rospkg, numpy, scipy, OpenCV, PIL, torchvision and PyYaml. Installation can either be through a docker image, or by building from the repository.

Usage

Because DOPE already has ROS integration, all it takes to test it is to start the ROS master, run a camera node like RealSense and start the DOPE node. Some rewriting of the configuration file may be necessary for it to work properly.

Training

The code for training is included, but it is not officially supported.

C.4 DeepIM

Software DeepIM runs on Python 2.7, but the authors note that is should be possible to use 3.x. Unlike the other methods, DeepIM requires OpenGL. Therefore it is necessary to install libglfw3-dev and libglfw3, and of course have OpenGL installed. Following that there is a few Python packages required that should be easily installed by pip. The authors also recommend git cloning of a package named Glumpy. The network used by DeepIM is also available as a python package, and as usual the actual files of DeepIM must be clone from the repository. The code has been tested on Ubuntu 14.04/16.04.

Usage Initialising DeepIM may take a while if following the recommendations of the authors. Running the init.sh file that readies and organises the data may take a few hours. This is just for the repository from github, using DeepIM with a custom-made dataset would require a new dataset file to be written akin to LM6D_REFINE.py.

Training DeepIM requires quite a few things to train on a new dataset. RGB images, ground truth poses, 3D models, camera intrinsic matrix and segmentation masks. To train on a custom-made dataset is, from what can be gathered from the discussions on the github repository[4], there is currently no official way of creating a dataset for neither DeepIM nor PoseCNN.

C.5 DenseFusion

Software

Installing DenseFusion is done through git cloning. The code works with Python 2.7/3.5/3.6, although 2.7 needs a rebuild of lib/knn/. The method has one branch for PyTorch 0.4.1 that works with Python 2.7, and one with PyTorch 1.0. At great cost to training speed, it is possible to run on CPU-only. However, CUDA 7.5/8.0/9.0 is highly recommended. Other requirements are PIL, scipy, numpy, PyYaml, logging and matplotlib. One thing that came up during testing was that torchvision had to be installed even when using PyTorch 0.4.1. By downloading torchvision the newer PyTorch 1.0 is installed along with it. To circumvent this it is necessary to force download the specific versions required.

Usage

As with SSP, the code uploaded to GitHub does not provide an easy to use command that allows for pose estimation in real time. There is ROS integration for a robot grasping

[4] `github.com/liyi14/mx-DeepIM` and `github.com/yuxng/PoseCNN`

experiment, but that code is not included in the GitHub repository. Therefore, it is necessary to create a new dataset and run it through the evaluation function that is provided. Simply testing DenseFusion is straight forward when all prerequisites are met. E.g. when testing on the LINEMOD dataset it is only necessary to run one script to download, and then one script to test. Testing and training on the YCB dataset is similar, but requires much more space as well as much more time to train.

Labels

The labels that DenseFusion use differ from the other methods. There are however a few tools that are recommended by Wang et al.[20]. One of these is LabelFusion[51], a tool specifically made for labelling RGB-D data in cluttered scenes. LabelFusions run in a Nvidia docker 1 container, and allows for mass labelling of data with relative ease. From ten seconds of filming, about 300 labelled images where created. Since docker and the attached image allows for the whole process to be contained, only a depth camera is required to create new datasets. Although LabelFusion provides much of the required data, it is not sufficient enough to create a correctly formatted dataset.

C.6 Dex-Net

Dex-Net 1.0

A growing dataset of over 10 000 unique 3D object models and 2.5 million parallel-jaw grasps, and an associated algorithm to study the scaling effects of big data and cloud computation on robust grasp planning. This first version of Dex-Net used Multi-View Convolutional Neural Networks, a deep learning method for 3D object classification, as a similarity metric between objects. This was then run simultaneously on up to 1500 virtual machines using the Google Cloud Platform. Experiments suggested that prior data could speed up the robust grasp planning by a factor of 2 on average. The quality of the grasps also increased with the number of similar objects in the dataset. The code for this is deprecated as of May 2017 [22, 52]. This gave a foundation for the rest of the project, as they now had a good dataset that they could use to build and expand upon.

Dex-Net 2.0

In an effort to reduce data collection times for deep learning in regards to robust robotic grasp plans, the Dex-Net team explored training from a synthetic dataset of 6.7 million point clouds, grasps and robust analytic grasp metrics. This was generated from the 3D models in Dex-Net 1.0, and the result was Dex-Net 2.0. The dataset is used to train a Grasp Quality Convolutional Neural Network (GQ-CNN) that rapidly predicts the probability of success of grasp from depth images. The grasps are specified with as the planar position, angle and depth of a gripper relative to an RGB-D sensor. The results from over 1000 trials suggest that a GQ-CNN can plan grasps in 0.8s with a success rate of 93% when it has been trained on synthetic data from Dex-Net 2.0, when planning grasps on eight known objects. When tested on ten common household objects, it achieved a 100% precision on 29 grasps. [22, 53]

Dex-Net 3.0

Suction-based end effectors are a widely used end effector in the industry, much due to their ability to lift objects with a single point of contact. It is also the solution used by Pickr.ai. Given that the method only requires one spot on a planar surface , the grasp planning becomes much easier than that of the parallel-jaw and multifinger grippers. A model that computes the quality of the seal between the suction cup and the target object is used to generate Dex-Net 3.0. To evaluate a given grasp, they measure the robustness to perturbations in the end-effector, as well as the object pose, material properties and external wrenches[5]. As before, a GQ-CNN is trained on the dataset. When the shape, pose and mass properties of an object is known, a 99% precision is achieved on objects of adversarial geometry [6]. A policy trained Dex-Net 3.0 achieves 99% and 97% precision respectively on a dataset of basic and typical objects. [22, 54]

[5]E.g gravity

[6]E.g sharply curved surfaces

Dex-Net 4.0

With the bin picking problem in mind, the Dex-Net 4.0 policy allows a physical robot to consistently clear a bin with 25 novel objects. This is done with a reliability greater than 95% at a picking rate averaging on 300 picks per hour. These policies are trained for both parallel-jaw and vacuum-based suction cup grippers, on 5 million synthetic depth images, grasps and rewards generated from heaps of 3D objects. [22, 48]

Bibliography

[1] DharaCNBC. Robots: The new low-cost worker, Apr 2015. URL https://www.cnbc.com/2015/04/10/robots-the-new-low-cost-worker.html.

[2] MHI. Industrial robots, unkown year. URL http://www.mhi.org/fundamentals/robots.

[3] Nick Statt. Amazon says fully automated shipping warehouses are at least a decade away, May 2019. URL https://www.theverge.com/2019/5/1/18526092/amazon-warehouse-robotics-automation-ai-10-years-away.

[4] Eirik Helland Urke. Unik video: Bli med inn i kompletts splitter nye robotlager, Jun 2015. URL https://www.tu.no/artikler/unik-video-bli-med-inn-i-kompletts-splitter-nye-robotlager/223905.

[5] Amazon. Prime air, unkown year. URL https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[6] Pickr.ai, 2019. URL https://www.pickr.ai/.

[7] Tanya M Anandan. Robotic bin picking - the holy grail in sight, Mar 2016. URL https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Robotic-Bin-Picking-The-Holy-Grail-in-Sight/content_id/6002.

[8] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. CoRR, abs/1804.00175, 2018. URL http://arxiv.org/abs/1804.00175.

[9] Zelin Zhao, Gao Peng, Haoyu Wang, Haoshu Fang, Chengkun Li, and Cewu Lu. Estimating 6d pose from localizing designated surface keypoints. CoRR, abs/1812.01387, 2018. URL http://arxiv.org/abs/1812.01387.

[10] Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao, and Xiaowei Zhou. Pvnet: Pixel-wise voting network for 6dof pose estimation. CoRR, abs/1812.11788, 2018. URL http://arxiv.org/abs/1812.11788.

[11] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. CoRR, abs/1612.08242, 2016. URL http://arxiv.org/abs/1612.08242.

[12] Autostore, 2019. URL https://autostoresystem.com/.

[13] James Vincent. Welcome to the automated warehouse of the future, May 2018. URL https://www.theverge.com/2018/5/8/17331250/automated-warehouses-jobs-ocado-andover-amazon.

[14] Eric Truebenbach. Is automated bin picking finally real?, Mar 2019. URL https://blog.universal-robots.com/is-automated-bin-picking-finally-real.

[15] ABB. Irb 360 - industrial robots (robotics) - industrial robots from abb robotics, 2019. URL https://new.abb.com/products/robotics/industrial-robots/irb-360.

[16] RobotWorx. Industrial robot integration, 2019. URL https://www.robots.com/robots.

[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

[18] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. CoRR, abs/1711.08848, 2017. URL http://arxiv.org/abs/1711.08848.

[19] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In Conference on Robot Learning (CoRL), 2018. URL https://arxiv.org/abs/1809.10790.

[20] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. CoRR, abs/1901.04780, 2019. URL http://arxiv.org/abs/1901.04780.

[21] Guoguang Du, Kai Wang, and Shiguo Lian. Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review. CoRR, abs/1905.06658, 2019. URL http://arxiv.org/abs/1905.06658.

[22] Jeffrey Mahler, Matt Matl, Bill DeRose, Jacky Liang, Alan Li, Vishal Satish, Mike Danielczuk, and Ken Goldberg. Dex-net, Jan 2019. URL https://berkeleyautomation.github.io/dex-net/.

[23] Canonical. Ubuntu releases, 2019. URL https://wiki.ubuntu.com/Releases.

[24] the python package index, 2019. URL https://pypi.org/.

[25] Python. Welcome to python.org, 2019. URL https://www.python.org/.

[26] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. Journal of Robotics, 2012:1–15, 2012. doi: 10.1155/2012/959013. URL http://dx.doi.org/10.1155/2012/959013.

[27] TullyFoote. Ros wiki, Nov 2018. URL http://wiki.ros.org/Documentation.

[28] Skymind. A beginner's guide to neural networks and deep learning, 2019. URL https://skymind.ai/wiki/neural-network.

[29] Skymind. A beginner's guide to convolutional neural networks (cnns), 2019. URL https://skymind.ai/wiki/convolutional-network.

[30] Niklas Donges. Pros and cons of neural networks, Apr 2018. URL https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b.

[31] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning: With Applications in R. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370. doi: 10.1007/978-1-4614-7138-7.

[32] Piotr Skalski. Preventing deep neural network from overfitting, Sep 2018. URL https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a.

[33] Lisa Torrey and Jude Shavlik. Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pages 242–264. Information Science Reference - Imprint of: IGI Publishing, 2009. ISBN 1605667668, 9781605667669. doi: 10.4018/978-1-60566-766-9.

[34] NVIDIA. Nvidia developer, 2019. URL https://developer.nvidia.com/.

[35] Admin. Overview of the intel realsense depth camera, Sep 2017. URL https://software.intel.com/en-us/realsense/d400.

[36] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692.

[37] Wikipedia. Random sample consensus, May 2019. URL https://en.wikipedia.org/wiki/Random_sample_consensus.

[38] David G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110, Nov 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[39] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In European conference on computer vision, pages 404–417. Springer, 2006. doi: 10.1007/11744023_32. URL https://doi.org/10.1007/11744023_32.

[40] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In Proc. Asian Conf. Computer Vision, volume 7724, 10 2012. doi: 10.1007/978-3-642-33885-4_60. URL https://doi.org/10.1007/978-3-642-33885-4_60.

[41] Papers with code : 6d pose estimation, 2019. URL https://paperswithcode.com/task/6d-pose-estimation.

[42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015. URL http://arxiv.org/abs/1512.02325.

[43] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[44] Ross B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

[45] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

[46] Jonathan Tremblay, Thang To, and Stan Birchfield. Falling things: A synthetic dataset for 3d object detection and pose estimation. CoRR, abs/1804.06534, 2018. URL http://arxiv.org/abs/1804.06534.

[47] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. CoRR, abs/1711.00199, 2017. URL http://arxiv.org/abs/1711.00199.

[48] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. Science Robotics, 4(26), 2019. doi: 10.1126/scirobotics.aau4984. URL https://robotics.sciencemag.org/content/4/26/eaau4984.

[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 2013 IEEE Conference on Computer Vision and Pattern Recognition, pages 2930–2937, June 2013. doi: 10.1109/CVPR.2013.377. URL doi.org710.1109/CVPR.2013.377.

[50] OpenCV Dev team. Introduction to surf (speeded-up robust features), Nov 2014. URL https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.

[51] Pat Marion, Peter R. Florence, Lucas Manuelli, and Russ Tedrake. A pipeline for generating ground truth labels for real RGBD data of cluttered scenes. CoRR, abs/1707.04796, 2017. URL http://arxiv.org/abs/1707.04796.

[52] Jeffrey Mahler, Florian T. Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James J. Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In Danica Kragic, Antonio Bicchi, and Alessandro De Luca, editors, 2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016, pages 1957–1964. IEEE, 2016. ISBN 978-1-4673-8026-3. doi: 10.1109/ICRA.2016.7487342. URL https://doi.org/10.1109/ICRA.2016.7487342.

[53] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. CoRR, abs/1703.09312, 2017. URL http://arxiv.org/abs/1703.09312.

[54] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David V. Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. CoRR, abs/1709.06670, 2017. URL http://arxiv.org/abs/1709.06670.