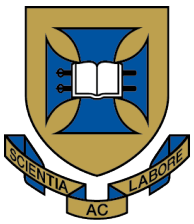


THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

A perception pipeline exploiting trademark databases for service robots

Joshua Song

B.E. (Hons)

A thesis submitted for the degree of Master of Philosophy at

The University of Queensland in 2019

School of Information Technology and Electrical Engineering

Abstract

Service robots are a potentially useful aid for elderly people or those with impairments. An example of a task that a service robot might be required to complete is object fetching; i.e. retrieving an object and bringing it to the person. However, the ability to recognize the many household objects a robot may encounter remains an open problem; this problem is the focus of this thesis. A MOVO robot was used for experiments, which is a mobile manipulator equipped with two 6-DOF (degrees of freedom) arms and a Kinect sensor. The first part of this thesis involved developing a perception pipeline for MOVO that processes raw sensor data into a format usable by the motion planner. The performance of several different object recognition algorithms was compared in a cup detection task. It was determined that a CNN (Convolutional Neural Network) outperformed other methods, but it was noted that it requires a significant number of training images.

Manually collecting information on all the objects a robot may encounter in a household is tedious and time-consuming. Therefore, the second part of this thesis examined the use of large-scale data from existing trademark databases. These databases contain logo images and a description of the goods and services the logo was registered under. For example, Pepsi is registered under soft drinks. In order to generate training data from the database images, RDSL (Randomization-based Data Synthesizer) was developed based on ideas from domain randomization. RDSL uses 3D rendering software to automatically generate synthetic data from the databases' logo images. A CNN logo detector trained on RDSL synthetic data outperformed previous logo detectors trained on synthetic data. The use of this logo detector was also demonstrated in a practical implementation for object fetching by MOVO. Tests on this robot indicated promising results, despite not using any manually-labelled real world photos for training.

Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

Publications included in this thesis

1. [1] **Joshua Song** and Hanna Kurniawati, Exploiting Trademark Databases for Robotic Object Fetching, *International Conference on Robotics and Automation*, 2019.

2. [2] Marcus Hoerger, **Joshua Song**, Hanna Kurniawati and Alberto Elfes, POMDP-based Candy Server: Lessons Learned from a Seven Day Demo, *International Conference on Automated Planning and Scheduling (ICAPS)*, 2019.

3. [3] Aaron J. Snoswell, Vektor Dewanto, Marcus Hoerger, **Joshua Song**, Hanna Kurniawati and Surya P.N. Singh, A distributed, any-time robot architecture for robust manipulation, *Australian Conference on Robotics and Automation*, 2018.

Submitted manuscripts included in this thesis

No manuscripts submitted for publication

Other publications during candidature

No other publications.

Contributions by others to the thesis

Dr. Hanna Kurniawati has constributed substantially to the conception and design of this project.

Statement of parts of the thesis submitted to qualify for the award of another degree

No works submitted towards another degree have been included in this thesis.

Research involving human or animal subjects

No animal or human subjects were involved in this research.

Acknowledgments

I would like to thank Dr. Hanna Kurniawati for her invaluable guidance and insight throughout this project, and for providing access to the MOVO robot. Thanks also goes to Dr. Surya Singh and fellow students Aaron Snoswell, Vektor Dewanto and Marcus Hoerger for their collaboration on our ACRA paper. I also like to thank TrademarkVision for allowing access to their trademark database. My advisory committee Prof. Stephen Wilson and Prof. Ross McAree have also provided valuable feedback during my progress milestones. Finally I thank my friends and family for their support during my entire thesis.

Financial support

This research was supported by an Australian Government Research Training Program Scholarship.

Keywords

Service robots, computer vision, object detection, deep learning, logos.

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080104, Computer Vision, 40%
ANZSRC code: 080108, Neural, Evolutionary and Fuzzy Computation, 40%
ANZSRC code: 080101, Adaptive Agents and Intelligent Robotics, 20%

Fields of Research (FoR) Classification

FoR code: 0801, Artificial Intelligence and Image Processing, 100%

Contents

List of figures

List of tables

List of abbreviations and symbols

Abbreviations	
CNN	Convolutional neural network
DOF	Degrees of freedom
OMPL	Open motion planning library
PCL	Point cloud library
POMDP	Partially observable markov decision process
PRM	Probabilistic roadmap
RANSAC	Random sample consensus
RCNN	Region-CNN
RDSL	Randomization-based Data Synthesizer for Logos
ROS	Robot operating system
RRT	Rapidly-exploring random tree
SIFT	Scale-invariant feature transform
SSD	Single shot multiBox detector

Chapter 1

Introduction

Robotics has made significant progress within the last few decades, advancing from simple manufacturing to autonomous cars, museum tour guide robots and affordable robot toys. Another potentially disruptive application are service or assistive robots, which are defined as robots that can assist people at work or in their homes with daily tasks, especially people with disabilities or impairments. In 2016, a group of universities published a "Roadmap for U.S. Robotics" [8] identifying gaps between robotics use cases and the current state of the art. The roadmap states that improvements in sensing and perception are critical to progress in robotics, particularly for robots in unconstrained environments. More specifically, it is important for robots to be able to recognize objects from 3D range data and RGB images and to also determine the object's position from that information.

In order for a robot to effectively collaborate with a human, it needs the ability to recognize the many objects it may encounter in a household or workplace. Semantic understanding is also a useful ability for a robot, that is, the ability for a robot to recognize the category an object falls under, such as "food" or "soft drink". Existing object recognition methods typically rely on either pre-training a classifier on many images of the object [9, 10] or looking up a database containing 3D data of the object during run time [11]. However, collecting this data is a time consuming task. While category level (e.g. bottle, cup) classification is feasible, instance level (e.g. bottle of Pepsi or Coca-Cola, cup of Starbucks coffee) is more difficult since more data is required and the objects can be difficult to differentiate. There is work in progress on collecting 3D data for various objects, but this is a time consuming task and these datasets are currently incomplete [11, 12].

Many objects have a distinguishing feature, which is a trademark or logo. Organizations such as the World Intellectual Property Organization (WIPO) maintain databases of logos and the category the logo was registered under. This work explored a novel approach: exploit the availability of *structured data* from trademark databases to enable a robot to detect objects that contain a logo and determine the category the object falls under.

Convolutional Neural Networks (CNN) have achieved state of the art accuracy in many perception tasks, but requires a large amount of training data. Since many millions of logos exist, gathering and labelling training data manually for each logo would be impractical. Synthetic data is a promising

approach for overcoming this limitation, whereby a graphic simulator is used to generate labelled images [13]. In this work, a new method was developed called Randomization-based Data Synthesizer for Logos (RDSL), that converts the trademark images of logos into synthetic "camera images" that can be used for training a CNN. Unlike previous synthetic data methods that start with a relatively meaningful simulated scene (e.g., a table for robotic grasping [14] and road for self-driving cars [13]), in our problem, we do not have any such initial scene. In fact, the input images we have for the data generator are stand-alone logos, while in the real world, these logos are placed on various different objects.

The algorithms developed in this thesis were tested on MOVO (see Figure 1.1), which is a mobile manipulator recently developed by Kinova Robotics to be safe for operating collaboratively with people. The University of Queensland's Robotics Design Lab has recently developed a Partially Observable Markov Decision Process (POMDP) system for robust grasping. This system was used in a candy scooping demonstration at the IEEE SIMPAR and ICRA 2018 conferences. A part of this thesis was developing the perception pipeline required for that task. This pipeline was then re-purposed for the logo-based object fetching trials.

The rest of this thesis is divided into three chapters. Chapter 2 provides a review of relevant literature. Chapter 3 describes the perception pipeline that was developed for MOVO under the Robot Operating System (ROS) framework. This chapter also compares the performance of a range of different object recognition techniques in a cup detection task under conditions that may be faced by a service robot. The most important and interesting contribution of this thesis, which is demonstrating an approach to exploiting trademarks for object recognition by service robots, is explained in Chapter 4. Here RDSL is explained, benchmarked against prior work, and shown to enable MOVO to complete an object fetching task in several scenarios without any manually labelled training data.

Figure 1.1: MOVO, a mobile manipulator by Kinova Robotics

Chapter 2

Background and Related Work

2.1 Service Robots

According to the international standard definition ISO 8373, a service robot is one that performs useful tasks for humans or equipment excluding industrial automation applications [15]. The level of autonomy can be partial (including human robot interaction) or full (without active human robot intervention). The global value of service robot sales in 2017 is estimated to be 6.6 billion dollars, with the primary drivers of growth including logistic robots (e.g. warehouse delivery), public relation robots (e.g. telepresence or mobile guidance) and vacuum robots [16]. Numerous research projects are focused on household "butler" robots for handicap assistance or elderly care. A total of 6,423 such robots were sold in 2017, up by 21% compared to 2016; this market is expected to increase substantially within the next 20 years [16].

One example of a service robotics research project is HERB [17], which is a mobile manipulator with one arm and two cameras. Due to the computational requirements of the image processing module, the images from the camera were streamed to three offboard computers for processing. HERB used a dataset containing 3D data and SIFT features for each object it needed to manipulate. The more recent Care-O-bot 3 [18, 19] provided a smartphone user interface that allows users to command the robot to fetch an object from a certain location. A RGB-D camera was used for object detection. A human tutor places new objects into the Care-o-bot's gripper, and the robot translates and rotates the object in order to learns 3D feature descriptors. The PR2 [20] is a robotics research platform that has enabled better research sharing and reproducibility by running Robot Operating System (ROS), an open-source software framework.

The Kinova MOVO beta (shown in Figure 1.1) is a mobile manipulator targeted towards use by researchers [21]. It is equipped with two lightweight arms (either 6-DOF or 7-DOF) that are safe for operation next to humans and is propelled by a holonomic wheeled base. MOVO is equipped with front and rear linear LIDAR sensors for navigation and mapping. It is also equipped with a Kinect V2 sensor mounted on a pan and tilt actuator, which is used for object detection.

(a) HERB [17] (b) Care-O-bot 3 [18] (c) PR2 [20]

Figure 2.1: Examples of service robots used for research

2.2 Robot Operating System

Robot Operating System (ROS) [22] is an open-source software framework designed for supporting collaboration and code re-usability in robotics research and development. ROS primarily supports Ubuntu Linux, although variants for Fedora Linux, macOS, and Microsoft Windows are available. ROS provides process management, message passing, hardware abstraction, and more. ROS libraries exist for many languages including C++, Python and Lua. In ROS, functionality is typically organized under separate processes called nodes. A node written in one language is still capable of communicating with a node written in a different language through data structures called messages. For example, a point message is defined as three floating point numbers, and a pose message is defined as a message containing both a point and a quaternion. Data can be exchanged asynchronously through a topic or synchronously via a service. In the topic model, a node can continuously publish messages which are held in a queue to be processed the subscriber(s). In the service model, a server waits for a request before sending a reply.

A typical pipeline under ROS might consist of the following nodes:

1. A sensor node that for example publishes laser scan data.

2. A perception node that processes the sensor data and publishes pose information of detected objects.

3. A high level motion planner that uses the information from the perception node to plan a trajectory in order to complete the task.

4. A low level actuator interface node that sends commands to the motors.

2.3 Motion Planning

Motion planning involves finding the solution for moving a robot from one state to another while respecting constraints and avoiding obstacles. The state space gets very large and computationally intractable with increasing degrees of freedom. MOVO has three degrees of freedom for its base (x, y and yaw), a movable torso, and two 6-DOF arms. This is a total of 16 degrees of freedom, although in this thesis motion planning was done separately for the base and the arm. Sampling-based motion planning is often required in order to quickly calculate a solution. The probabilistic roadmap (PRM) [23] is one of the first sampling-based algorithms and involves randomly sampling from the state space and connecting nearby states to form a roadmap. Once the roadmap is complete a path connecting the start and goal states can be found through a graph search algorithm such as A* [24]. Rapidly-exploring random tree (RRT) [25] constructs a tree by repeatedly sampling a state and connecting it to the nearest state in the tree. RRTs are suitable for problems with non-holonomic or dynamic constraints (e.g. a car) as new states can be sampled by applying control inputs to existing states. Unlike PRM, tree-based planners are considered single-query planners as a new tree has to be constructed for each query.

For a robot arm, a goal is often specified as a pose (position and orientation) for the end-effector (i.e. the robot's hand). Forward kinematics is the mathematical process for calculating the end-effector position given the joint angles, while the reverse process is known as inverse kinematics.

The Open Motion Planning Library (OMPL) [26] is an open-source software library that contains PRM, RRT and many other sampling-based motion planning algorithms. This thesis utilized MoveIt! [27], which is a ROS library that wraps OMPL along with collision checking and kinematics solvers. A MoveIt! configuration for a robot can be constructed through an Unified Robot Description Format (URDF) file, which is a description of the robot's joints, joint limits and the links between the joints. MOVO comes already setup with a MoveIt! configuration.

Environments that a robot operates in can be unpredictable or unstructured. Imperfect actuators may result in errors and imperfect sensors means that the state of the environment is never known exactly. A robot that does not take uncertainty into account may use motion plans that are not robust or overly conservative and sub-optimal. A Partially Observable Markov Decision Process (POMDP) [28] offers a framework for operating under uncertainty. Rather than planning over states, a POMDP plans over beliefs, which are probability distributions over states. The robot operates in an act-observe cycle where it repeatedly performs an action and updates its belief from an observation. Because a POMDP plans over beliefs rather than states, solving POMDPs is even more computationally expensive compared to planners that do not account for uncertainty. Sampling-based methods, such as POMCP [29], DESPOT [30] and ABT [31], have steadily improved the performance of POMDP solvers and may now be usable in a wider range of robotics applications.

2.4 Object Detection Methods

One of the earliest approaches to object detection was Random sample consensus (RANSAC) [32]. RANSAC involves sampling a subset of points, fitting a model to that subset, and calculating how well the rest of the data fits that model. For example, in order to fit a circle to a set of 2D points, RANSAC samples three points, since a minimum of three points are required to fit a circle. In addition to 2D images, RANSAC has been applied to 3D point cloud data in order to fit planes, cylinders and other shapes [33].

In order to detect more complex objects than lines and circles, feature-based methods may be used. One such method is Scale-Invariant Feature Transform (SIFT) [34]. SIFT determines a description of a training image based on interesting points called keypoints or features. These features usually lie on high-contrast regions, such as edges. This feature-description can be stored in a database, and then compared to features extracted from a new image in order to detect the object. SIFT is invariant to orientation, uniform scaling, lighting changes, and partially invariant to affine distortion.

SIFT has been adapted to work with point cloud data [35]. Other feature-based methods for point cloud object detection include Viewpoint Feature Histogram (VFH) [36], Normal Aligned Radial Features (NARF) [37], and Signature of Histograms of OrienTations (SHOT) [4]. An example of SHOT is shown in Figure 2.2. These methods require a point cloud of the object, which can be obtained either by scanning the object with a sensor or by converting a CAD model of the object into a point cloud. The KIT [12] dataset contains 3D data of over 100 objects. Its setup used cameras and a high-accuracy laser scanner to scan an object placed on a turntable. The BigBIRD [11] dataset was constructed through a calibrated multi-camera system and is fully automated, meaning capturing data of an object can be done within 5 minutes. This dataset consists of 100 objects but is still being expanded.

More recently, Convolutional Neural Network (CNN) based methods have surpassed previous methods in many perception tasks. This is further covered in Section 2.5.

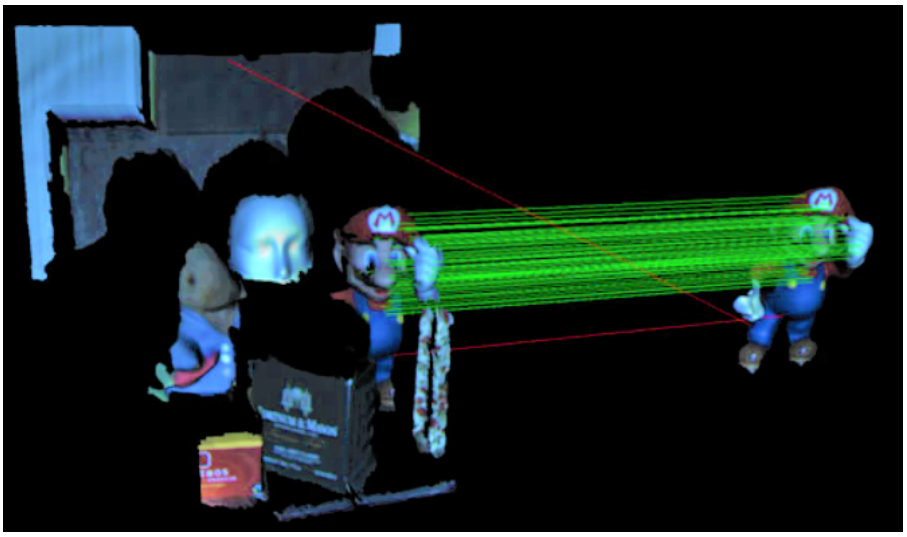


Figure 2.2: Object detection example using SHOT feature descriptors. Correct matches are shown as green lines while red lines are incorrect matches [4].

2.5 Convolutional Neural Networks

A standard neural network consists of connected processors called artificial neurons. Each neuron computes a weighted sum of its inputs and then passes a single value through a non-linear activation function. The parameters for the weighted sum and activation functions are commonly obtained through supervised learning, where the network is trained on a data set that has been labelled with the correct classes.

In a typical image classification network the first layer is the pixel values of an image, the hidden or middle layers of neurons perform computation, and the output layer outputs the probabilities for each class. Deep learning is a general term for networks with many layers. The layers are not limited to simple neurons but can also perform other computation, such as convolution.

Convolutional neural networks (CNN) are designed for processing array data, such as images. They have gained popularity since the 2010s when they were applied successfully to image processing tasks such as handwriting recognition and face recognition [38] and achieving state of the art performance in the ImageNet challenge, which involved classifying images into 1000 classes (such as “dog”, “cup”, “speedboat”, etc.) [39]. Instead of connecting each pixel of an image to a neuron, a CNN learns a set of filters, which commonly have sizes 3x3 or 5x5. These filters are then convolved on the image. This way, what is learned for one part of the image can be applied to other parts of the image. This is visualized in Figure 2.3. In the early layers, simple features such as edges and curves are learned, while in the later layers, more complex components such as wheels and windows are learned. In addition to convolution layers, common layers used in CNNs include pooling layers, which down-samples the data, and fully connected layers, which are neurons as seen in standard neural networks. The arrangement of the layers is still an active area of research. A few recently developed architectures include NASNet [40], ResNet [41], Inception [42] and MobileNet [43]. MobileNet is a smaller and faster network while still having similar accuracy to VGG.

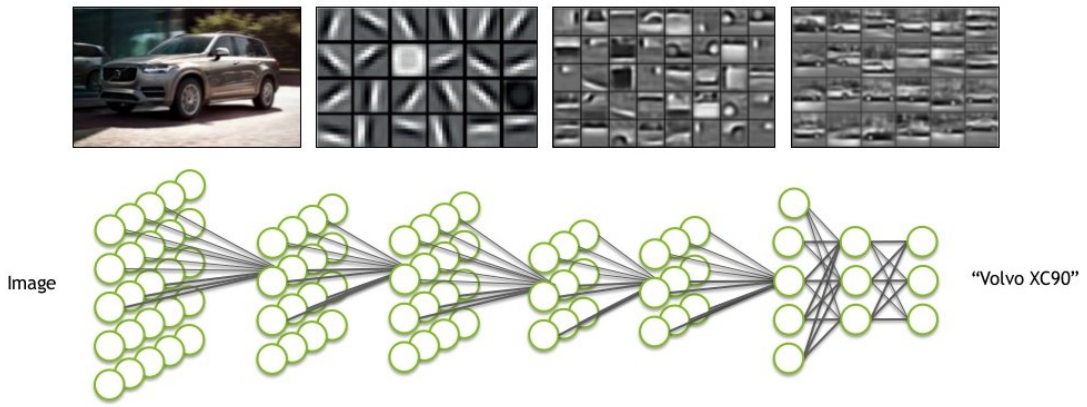


Figure 2.3: Simplified visualization of a convolutional neural network [5]

CNNs usually require a large amount of annotated data to perform well. Instead of training a new model from scratch, the time and data required for training can be reduced through a process called transfer learning or fine-tuning [44]. In this process, a model that has been trained for a certain task

is re-trained on data for the new task. The architecture of the model remains the same except for the output layer, which is changed to match the number of classes in the new task. Fine-tuning works as features learned on data from a certain task can often be re-used on another task.

In a classification configuration, CNNs are invariant with regard to the position of the object within the image. However, it is often useful to localize and identify multiple objects within the image. One of the latest works in object detection is You Only Look Once (YOLO) [45], a sample is shown in Figure 2.4. Another method is Regions with CNN features (R-CNN) [46], which first finds regions of interest based on color and texture, then classifies the regions through a CNN and Support Vector Machine (SVM). Detection speed was improved with Fast R-CNN [47] and Faster R-CNN [10]. Faster R-CNN uses convolution features for both region proposals and class prediction. An even faster method (though slightly less accurate) is Single Shot MultiBox Detector (SSD) [9], which eliminates proposal generation and performs all computation in a single network. These object detection methods extend other CNN architectures, for example, configurations may include Faster R-CNN with ResNet, Faster R-CNN with Inception, SSD with Inception, and so on.

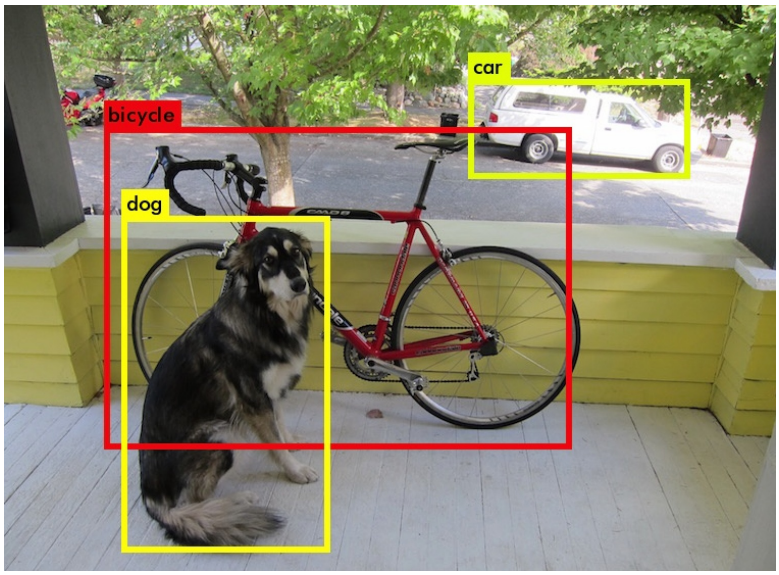


Figure 2.4: YOLO object detection [6]

The previously listed object detection networks are used for finding bounding boxes around objects. Semantic segmentation networks such as Fully Convolutional Network (FCN) [48] and Pyramid Scene Parsing Network (PSPNet) [49] assigns class labels to each pixel.

In addition to class labels, object detectors also require their training data to be annotated with either bounding boxes or pixel masks for each object instance. Publicly available datasets include COCO [50] and Open Images [51], which are datasets for 91 and 600 object categories, respectively. Ammirato et al. [52] modified general object detectors trained on such datasets to be able to detect specific object instances (e.g. my mug instead of any mug), given several new images of the object instance.

Instead of training a new CNN model from scratch, the time and data required for training can be reduced through a process called transfer learning or fine-tuning [44]. In this process, a model that has

been trained for a certain task is re-trained on data for the new task.

The training data can be expanded through augmentation techniques such as rotating, flipping and cropping images [53]. In addition to data augmentation, new training data can be generated through synthetic means such as simulators and domain randomization. Domain randomization involves randomizing the parameters of the simulator in unrealistic ways in order to force the CNN to learn the essential features of the target object. This technique was used by [14] to train an object detector for robot localization and grasping. They suggest that "with enough variability in the simulator, the real world may appear to the model as just another variation". *Flying distractors* was introduced by [13] as a domain randomization component, which are random geometric shapes added to the scene. They then demonstrated the effectiveness of domain randomization for car detection.

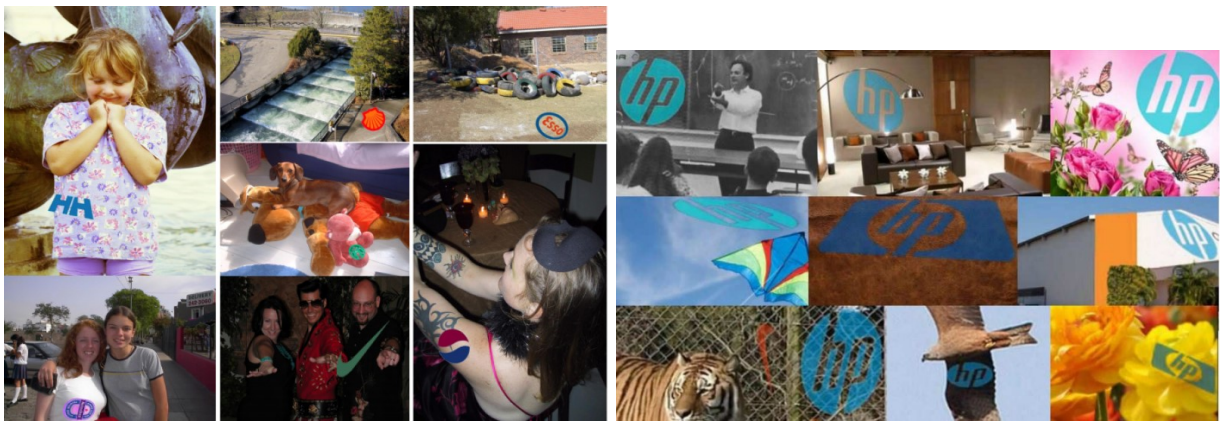
Hinterstoisser et al. [54] generated synthetic data for a household objects detector by composing CAD models of those objects on random background images. They found that freezing the feature extractor layers and only fine-tuning the region proposal layers improved performance.

Similar to the previous methods, our synthetic data generator, RDSL, also uses randomization. However, unlike those methods that start with a relatively meaningful simulated scene (e.g., a table for [14] and road for [13]), in our problem, we do not have any such initial scene. In fact, the input images we have for the data generator are stand-alone logos, while in the real world, these logos are placed on various different objects.

2.6 Logo Detection

Early approaches to logo detection used keypoint detectors, such as SIFT [55, 56]. More recent work has shown that CNNs outperform previous methods for logo recognition [57–59]. The improvements in computer vision have been employed by several companies in order to provide query-by-image services for logos. For example, TrademarkVision [60] allows a user to upload a logo image in order to find similar logos for the purpose of finding copyright infringement. LogoGrab [61] provides a logo detection service for monitoring social media and counterfeit product detection.

Eggert et al. [58] and Su et al. [59] generated synthetic data for logos by applying random geometric and illumination transforms to logos and pasting them on random photos as shown in Figure 2.5a. Montserrat et al. [62] followed a similar approach but used a CNN to estimate depth in the background image in order to blend the logo image more realistically as shown in Figure 2.5b. Rather than overlaying logos on random photos, RDSL generates synthetic images by rendering the logos on randomized geometric shapes, which is more suitable for this robotics task where the goal is to detect logos on relatively small objects of varying shapes.



(a) Samples from [59] (b) Samples from [62]

Figure 2.5: Samples of synthetic training data for logos from previous work

The following publications have been incorporated as Chapter 3.

[3] Aaron J. Snoswell, Vektor Dewanto, Marcus Hoerger, **Joshua Song**, Hanna Kurniawati and Surya P.N. Singh, A distributed, any-time robot architecture for robust manipulation, *Australian Conference on Robotics and Automation (ACRA)*, 2018.

[2] Marcus Hoerger, **Joshua Song**, Hanna Kurniawati and Alberto Elfes, POMDP-based Candy Server: Lessons Learned from a Seven Day Demo, *International Conference on Automated Planning and Scheduling (ICAPS)*, 2019.

Contributor	Statement of contribution	%
Marcus Hoerger	writing of text	50
	theoretical derivations	60
	preparation of tables, figures	70
	pomdp planner coding	100
	robot trials and experiments	50
	initial concept	20
Joshua Song	writing of text	20
	preparation of tables, figures	30
	perception pipeline coding	100
	robot trials and experiments	30
	initial concept	20
Hanna Kurniawati	writing of text	30
	theoretical derivations	40
	initial concept	60
	robot trials and experiments	20
	supervision, guidance	60
Alberto Elfes	supervision, guidance	40

Table 2.1: Contributions for ACRA paper

Contributor	Statement of contribution	%
Aaron J. Snoswell	writing of text	40
	preparation of tables, figures	60
	hardware configuration	70
	framework coding	50
	robot trials and experiments	25
	initial concept	10
Vektor Dewanto	writing of text	20
	preparation of tables, figures	20
	hardware configuration	30
	framework coding	50
	robot trials and experiments	25
	initial concept	10
Marcus Hoerger	writing of text	10
	theoretical derivations	60
	pomdp planner coding	100
	robot trials and experiments	25
	initial concept	10
Joshua Song	writing of text	10
	preparation of tables, figures	20
	perception pipeline coding	100
	robot trials and experiments	25
	initial concept	10
Hanna Kurniawati	writing of text	10
	theoretical derivations	40
	initial concept	50
	supervision, guidance	50
Surya P.N. Singh	writing of text	10
	initial concept	10
	supervision, guidance	50

Table 2.2: Contributions for ICAPS paper

Only the sections relevant to this thesis (i.e. the perception pipeline sections) are included in Chapter 3.

Chapter 3

Pipeline for Object Detection on the MOVO Robotic Platform

3.1 Introduction

The University of Queensland obtained the MOVO robot from Kinova Robotics as part of a beta program. MOVO is a mobile manipulator consisting of two 6-DOF (degrees of freedom) arms propelled by a holonomic wheeled base. MOVO is equipped with front and rear linear LIDAR sensors for navigation and mapping. It is also equipped with a Kinect V2 sensor mounted on a pan and tilt actuator, which is used for localizing objects for manipulation. The Kinect is a low-cost time-of-flight sensor that provides both RGB and depth images. ROS provides an interface that projects each pixel in the depth map into a point in 3D space, forming a point cloud, which is a set of 3D points.

We were tasked with developing a system architecture for demonstrating robust manipulation during the SIMPAR 2018 and ICRA 2018 conferences. The chosen demonstration was a candy scooping task shown in Figure 3.1. A box containing candy is placed on the floor next to MOVO. This box has a fiducial marker to ease its detection. A cup is placed on the table in front of MOVO. The table may also hold other objects which act as obstacles. MOVO needs to detect the cup, grasp it, use it to scoop candy from a nearby box, and finally place it back on the table without spilling any candy and without colliding with the table or other obstacles. The system achieved robust grasping through use of a Partially Observable Markov Decision Process (POMDP) framework and a state-of-the-art POMDP solver. In order to maintain a reasonable planning horizon and online update rate, processing was distributed across the onboard computers and a offboard planning server. Robot Operating System (ROS) was used to facilitate data communication. My sole contribution was developing the perception or computer vision algorithms and the pipeline for interfacing with the rest of the system.

Figure 3.1: Candy scooping demonstration setup

Section 3.2 provides the implementation details. The pipeline for processing sensor data and then passing observations to the POMDP planner is described in Subsection 3.2.1. The rest of Section 3.2 describes the perception algorithm, which consisted of three main components: tabletop segmenter, pose tracker and convolutional neural network (CNN). The tabletop segmenter, described in Subsection 3.2.2 splits the point cloud from the kinect into clusters corresponding to the table and objects placed on top of the table. The pose tracker, described in Subsection 3.2.3 extracts the pose (position and orientation) of the cup from the point cloud. This pose tracker requires an initial position estimate of possible cup locations. Once it has "locked on" to the cup, the pose tracker can continuously process updated point clouds to track the cup as it is moved. In an early implementation, the object clusters from the tabletop segmenter were used for providing the initial cup position estimates, but a later implementation used a CNN, which is described in Subsection 3.2.4. The CNN provided a more reliable position estimate of the cup, especially in scenes with significant clutter. This position estimate was passed to the pose tracker, which could determine the pose of the cup more precisely. If the pose tracker loses track of the the cup (e.g. if the cup is moved too fast), the CNN can detect the cup again. Feature descriptor based methods (e.g. SIFT) were also explored, but were found to produce poorer results. These are described in Subsection 3.2.5.

The results from tests are shown in Section 3.3. Subsection 3.3.1 compares the detection algorithms in terms of whether or not they can detect the cup in the presence of similar objects or clutter while Subsection 3.3.2 analyzes the localization accuracy, i.e. how accurate the estimated pose is compared to the true pose.

3.2 Method

3.2.1 System Overview

Our distributed system shown in Figure 3.2. It consists of two Intel NUC mini PCs onboard MOVO, and an external desktop PC. The specifications of these computers are tabulated in Table 3.1. The POMDP planner was run on an external PC as it required a different ROS version than the one on used on MOVO, and due to time constraints the ROS version on MOVO could not be updated. The nodes for processing the data from the Kinect sensor are located on the MOVO2 PC, and consume approximately 40% of the CPU's processing capability.

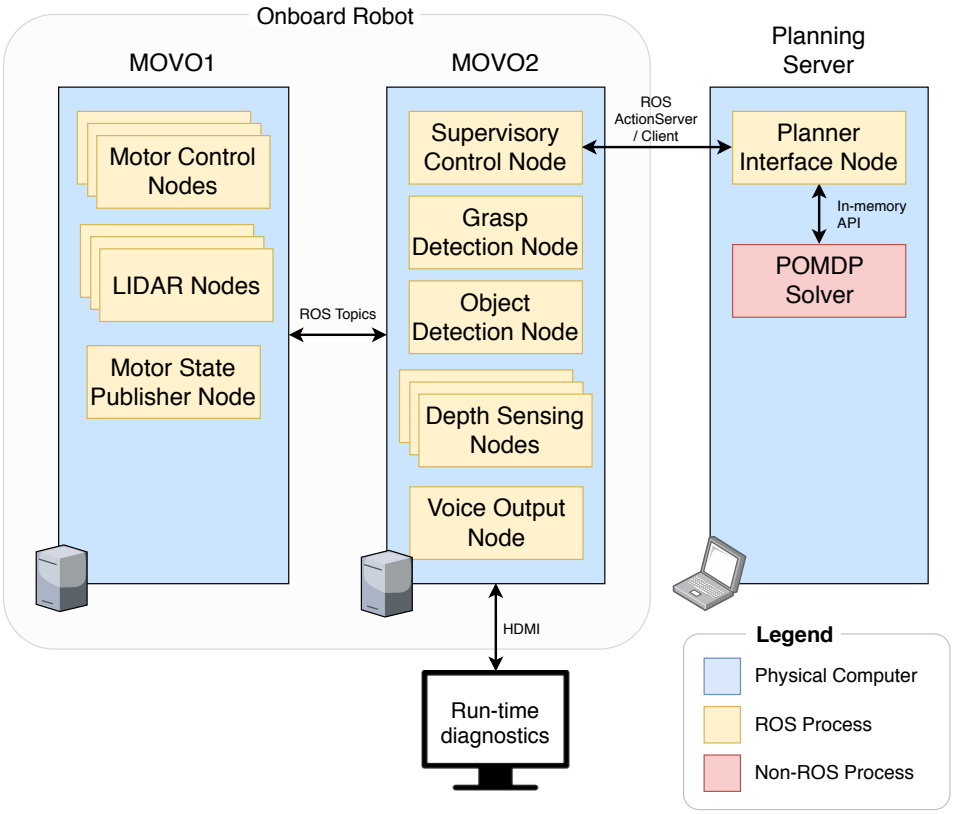


Figure 3.2: Distributed architecture used in our system [3]

Table 3.1: Specifications for the computers used in our system architecture

Computer	MOVO1	MOVO2	Planning Server
Model	Intel NUC5i7RYB	Intel NUC5i7RYB	Dell Precision T3600
Processor(s)	Intel Core i7-5557U @ 3.10GHz (2 cores)	Intel Core i7-5557U @ 3.10GHz (2 cores)	Intel Xeon E5-1620 @ 3.60GHz (4 cores)
Operating System	Ubuntu 14.04.5 64-bit LTS + real time kernel	Ubuntu 14.04.5 64-bit LTS + real time kernel	Ubuntu 16.04.1 64-bit
Disk Space	128GB	128GB	1TB
Memory	16GB DDR3 1867MHz	16GB DDR3 1867MHz	16GB DDR3 1600MHz
ROS Version	8 (Indigo Igloo)	8 (Indigo Igloo)	10 (Kinetic Kame)

A flowchart of the process used for the entire candy scooping process is shown in Figure 3.3. When MOVO is powered on, it runs a self-diagnostic test by stretching its arms to a preset position to ensure they are working correctly. MOVO then performs a visual scan. First, the arms are moved to the side so that the Kinect sensor is not obstructed, then MOVO pans the Kinect in order to find the candy container and determine the container's pose through a fiducial marker. The fiducial marker used is from the ROS Alvar package and is shown in Figure 3.4. The height of the candy was then measured by averaging the height of the relevant points in the point cloud. The table and cup is then localized and this information forms the initial observation that is passed to the POMDP solver. During runtime, the POMDP solver operates in an act-observe loop, where an observation is only needed at the end of an action. Rather than running continuously and wasting CPU resources, the perception nodes only run when a request is received from the POMDP planner through a ROS service, instead of continuously publishing observations on a topic.

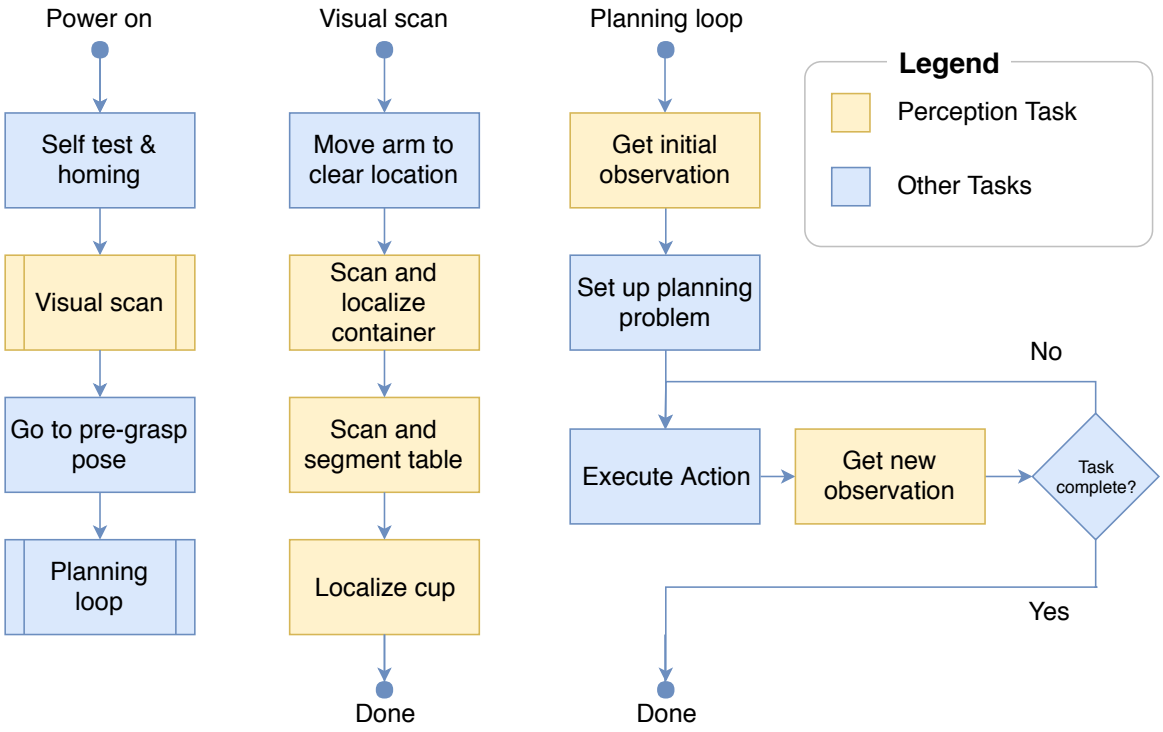


Figure 3.3: Candy scooping process flowchart

Figure 3.4: Fiducial marker used for localizing the candy container [7]

3.2.2 Point Cloud Preprocessing and Tabletop Segmenter

The tabletop segmenter is responsible for extracting bounding boxes for the table and objects located on the table from the input point cloud. A flowchart showing where the tabletop segmenter interfaces with the rest of the system is shown in Figure 3.5. The input point cloud from the Kinect is quite noisy, and must be preprocessed for better results. This preprocessing step is explained further in Figure 3.6. The tabletop segmenter then splits the preprocessed point cloud into clouds corresponding to the table and each object on the table, and calculates the bounding boxes for each cloud as shown in Figure 3.7. The pose tracker differentiates the cup cloud from the other object clouds; this process is described in Subsection 3.2.3. Note that this flowchart shows the early version without the CNN component. The final system with the CNN component is described in Subsection 3.2.4.

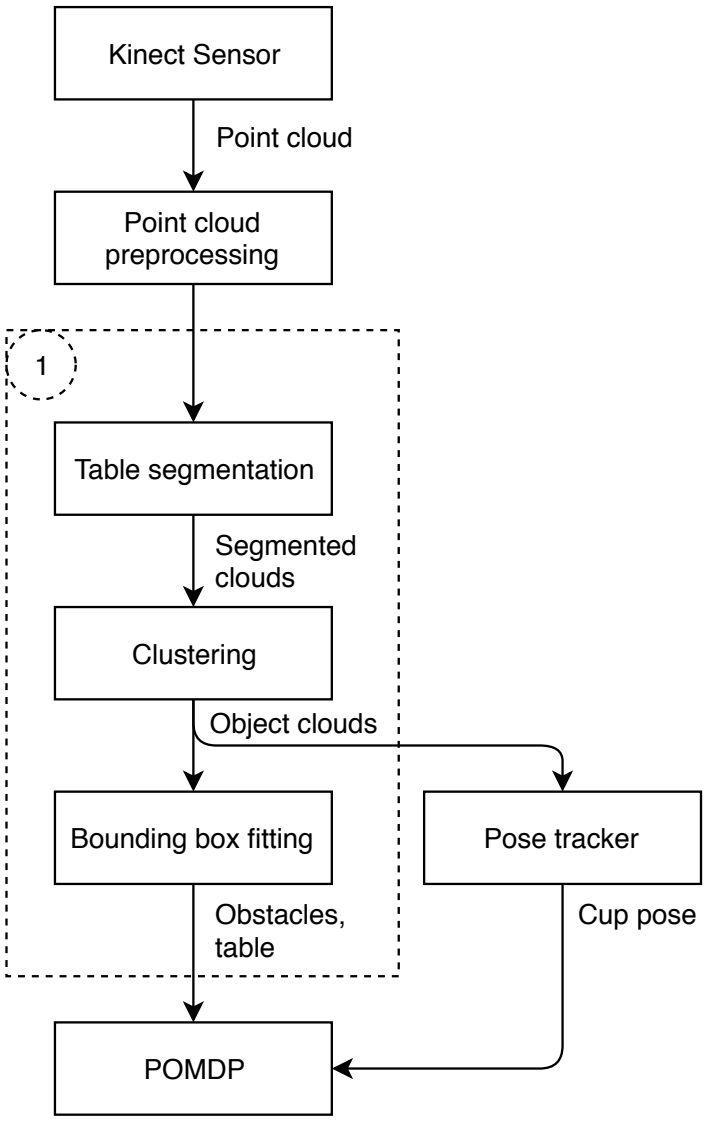
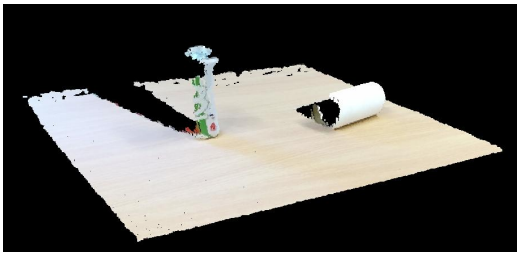


Figure 3.5: Tabletop segmenter interface flowchart. The tabletop segmenter component is marked by the dashed box.

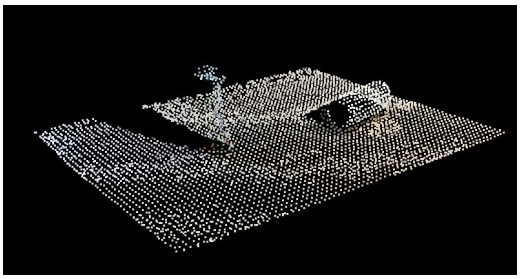
(a) Point cloud after cropping

Functions from the Point Cloud Library (PCL) [35] were used for manipulating the point cloud. Point cloud preprocessing involves first cropping the point cloud to within the workspace. Due to sharp curvature and material reflectivity, there are some erroneous points, particularly around the soft drink can.



(b) Statistical outlier filter

A statistical outlier filter is applied in order to remove these erroneous points. This filter first computes the average distance of each point to its nearest k neighbors (here k=35 was used). The mean and standard deviation of the distances are then computed. Points that have an average neighbour distance of more than 0.9 standard deviations from the mean are discarded.

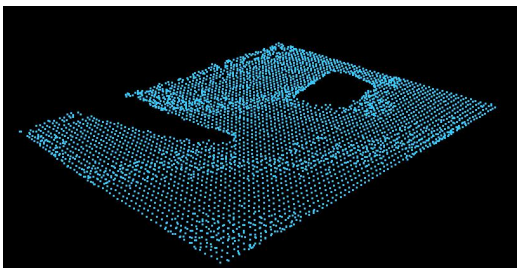


(c) Voxel grid filter

A voxel grid filter is applied to down-sample the data to reduce computation in later steps. A voxel size of 1 cm was used.

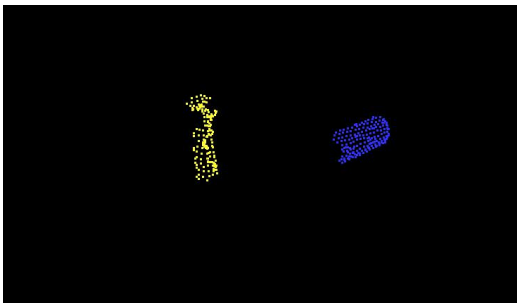
Figure 3.6: Point cloud preprocessing steps

The final step in the point cloud preprocessing stage is a robot self-filter. If the robot's arm is in the scene, the points corresponding to the arm can be removed using the robot arm's geometry and joint angles. The now preprocessed cloud is then passed to the tabletop segmenter.



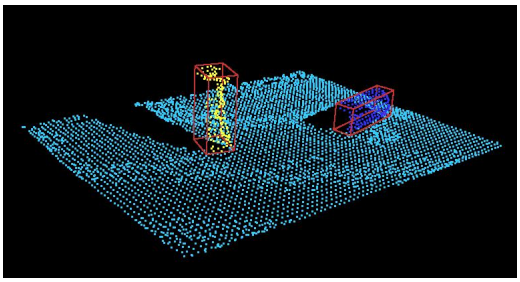
(a) Plane segmentation

The table surface points are segmented using the RANSAC algorithm [32].



(b) Clustering

A clustering algorithm is then run on the non-table points. This algorithm clusters points that are within a certain euclidean distance of each other (in this case 2 cm).



(c) Object bounding boxes

The eigenvalues and eigenvectors of the covariance matrix of each cluster are computed in order to find the major and minor axes. This allows an oriented bounding box to be computed. The z component is not used in the calculation of the axes, so that the bounding box lies flat on the table.

Figure 3.7: Tabletop segmenter steps

3.2.3 Pose Tracker

Although the cup is stationary most of the time, it can be accidentally moved in a bad grasp manoeuvre. In this case, it is helpful to be able to continuously track the cup. The pose tracker is based on PCL's object tracking library [35], which uses a particle filter approach to determine the object's pose (position and orientation). A particle filter was used as it can provide a probabilistic estimate of a range of poses instead of a single best guess, which could be useful in future POMDP extensions. The library was modified to ensure that the object neither collides with the table nor floats above it, as this provided more reliable results. In addition to object tracking, the pose tracker is also responsible for differentiating the cup cluster from other clusters. A summary of the tracker algorithm is given in Pseudocode 1. The tracker is provided a 3D CAD model of the target object, which is loaded as a point cloud and down-sampled by a voxel grid filter to the same voxel size as in the point cloud preprocessing step (1 cm). The tracker is also provided an initial position estimate. At each tracking iteration, the tracker is updated with a new point cloud from the Kinect sensor as well as the plane corresponding to the table, which was segmented in the preprocessing stage. The plane is specified as a vector of coefficients (A,B,C,D), corresponding to the plane's function $Ax+By+Cz+D=0$

with $\sqrt{A^2+B^2+C^2} = 1$. A particle is defined as a pose (x, y, z, roll, pitch, yaw) and a weight value. The tracker's update function, which performs resampling to create a new set of particles, is called whenever a new point cloud is received during the grasping manoeuvre. A new particle is generated by sampling a particle from the previous set and adding Gaussian noise to its pose. The object cloud (derived from the CAD model) is then transformed to the new particle's pose. The distance between the lowest point in the object cloud and the plane is then calculated from the dot product, and the pose is shifted up or down to ensure that the object does not intersect with the table or float above it. The new particle's weight is determined by calculating the coherence between transformed object cloud and the input sensor cloud, which is the negative of the sum of distances between nearest point pairs. In the early configuration (where no CNN was used), the pose tracker was run using each cluster from the tabletop segmenter as the input cloud and the cluster's centroid as the initial position. The cluster that produced the highest weight (i.e. the most coherence) was labelled as the cup cluster, and the rest of the clusters were labelled as obstacles. In the final configuration (with the CNN described in Subsection 3.2.4), the entire point cloud from the Kinect sensor was used as the input cloud and the position estimate was provided by the CNN.

Pseudocode 1 Pose tracker

```
function INITIALIZE(objectCloud, initialPosition)
    Down-sample objectCloud with voxel grid filter
    p.pose.position ← initialPosition
    p.pose.orientation ← (0,0,0)
    p.weight ← 1
    particles ← empty set
    Add p to particles
    totalWeight ← 1
end function
function COMPUTECOHERENCE(cloud1, cloud2)
    dist ← 0
    for Particle p1 in cloud1 do
        p2 ← point in cloud2 closest to p1
        dist ← dist+ distance between p1 and p2
    end for
    return −dist
end function
function UPDATE(sensorCloud, tablePlane)
    newParticles ← empty set
    newTotalWeight ← 0
    bestWeight ← ∞
    for i ← 1 to numParticles do
        Sample particle oldP from particles with probability oldP.weight / totalWeight
        p.pose ← oldP.pose + Gaussian noise
        c ← transform objectCloud to p.pose
        lowestPoint ← (x,y,z,1) where x,y,z correspond to the lowest point in c
        dist ← lowestPoint · tablePlane
        p.pose.z ← p.pose.z - dist
        c ← transform objectCloud to p.pose
        p.weight ← COMPUTECOHERENCE(c, sensorCloud)
        Add p to newParticles
        newTotalWeight ← newTotalWeight + p.weight
        if p.weight > bestWeight then
            bestP ← p
            bestWeight ← p.weight
        end if
    end for
    particles ← newParticles
    totalWeight ← newTotalWeight
return bestP.pose, bestWeight
end function
```

3.2.4 Convolutional Neural Network for Cup Detection

The tabletop segmenter and pose tracker often fails to detect the cup correctly if there are similarly shaped objects or significant clutter on the table. To remedy this, a CNN was trained using the Tensorflow [63] framework. The architecture selected was SSDLite-MobileNetV2 [64], as recognition needed to be carried out quickly and on a computationally limited platform. The CNN was pre-trained on the COCO [50] dataset and then fine-tuned for 200k steps on 200 photos of the target object. These

photos were manually labelled with bounding boxes of the cup as shown in Figure 3.8.

During runtime, the CNN takes an image from the Kinect as input and outputs a bounding box of the logo in pixel coordinates, which must then be converted to world coordinates (i.e. XYZ coordinates in meters). This conversion can be done by using the distance measurement and the intrinsic and extrinsic matrices. ROS's Kinect interface already performs this calculation in order to convert the depth image from the Kinect into a point cloud, which is a set of XYZ points. Therefore, it is more convenient to extract the world coordinates by calculating the pixel's array index in the point cloud.

The final configuration, where the CNN is used instead of the tabletop segmenter as input to the pose tracker, is shown in Figure 3.9.

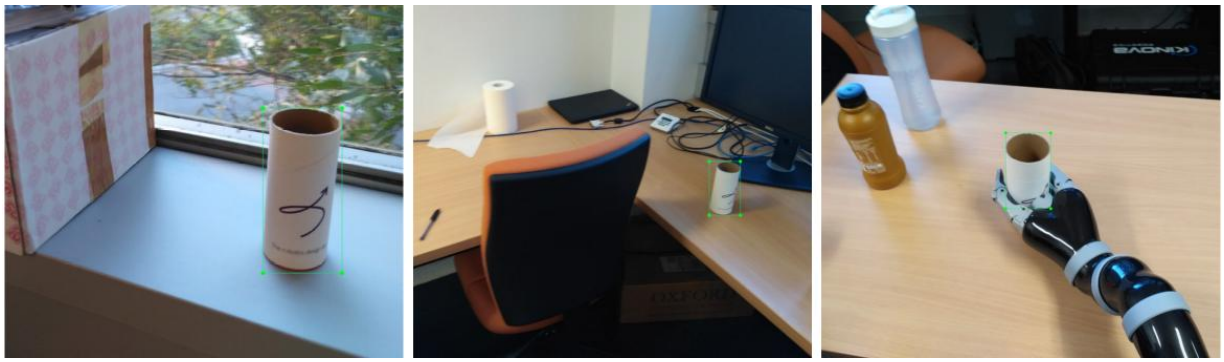


Figure 3.8: Samples of images labelled with bounding boxes used to train the cup CNN

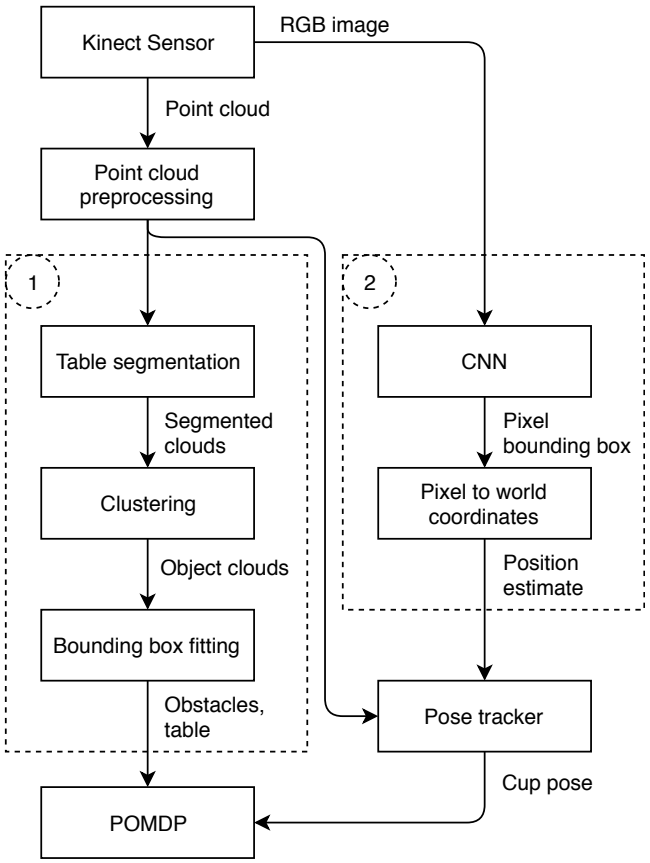


Figure 3.9: Final vision pipeline. Dashed box 1 is the tabletop segmenter component, dashed box 2 is the CNN component.

3.2.5 Feature Descriptor Methods

Feature descriptor based methods for detecting the cup were explored in two different approaches; first using the RGB image from the Kinect as input, and second, using the point cloud as input. The image feature descriptors used included SIFT [34], SURF [65] and ORB [66]. The implementations from the OpenCV library [67] were used. The same 200 images used for training the CNN were used for training the feature based methods. The images were cropped to the cup bounding boxes and then the features extracted. The length of the feature vector varies depending on the algorithm, being 128, 64 and 32 for SIFT, SURF and ORB, respectively. The total number of training features also varies depending on the algorithm and threshold settings, and were 3155, 3902 and 420.

During inference, features are extracted from the image from the Kinect and for each of these features, the closest matching feature from the training stage is found. A match is only kept if the distance between the feature vectors is below a certain threshold. The matches are then clustered together based on pixel distance, and the average pixel coordinates of each cluster is returned as a cup detection. An example is shown in Figure 3.10.

The second approach using point clouds is quite similar. It was done with PCL's implementation of SHOT [4] features. The training features were extracted from the 3D CAD model of the cup.

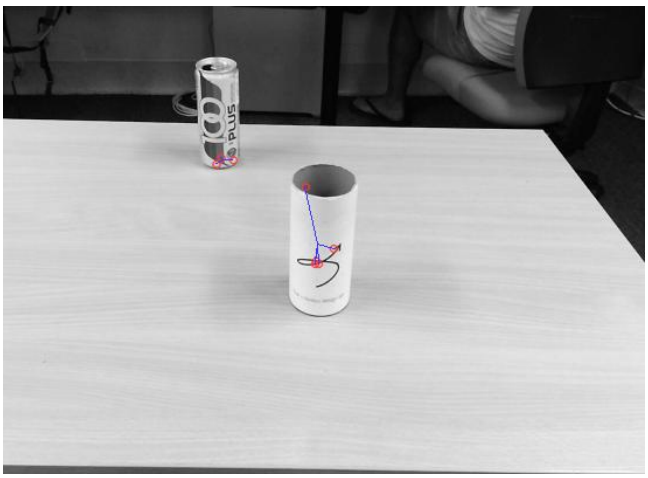


Figure 3.10: Sample detection using ORB. The red circles show matched features, the blue lines connect features in the same cluster. The 100PLUS can is a false positive.

3.3 Results

3.3.1 Comparison of Cup Detection Algorithms

For testing the cup detection algorithms, a number of test sets were constructed by placing the cup on the table along with different distractor objects and then recording both the RGB image and the point cloud from the Kinect. The tests in this section are more concerned about correctly identifying the cup and less about the pose accuracy. The ground truth cup pose was recorded by using the pose tracker algorithm before any distractors were placed. A detection was considered a true positive if it was within 5 cm of the ground truth. Sample photos of the test sets are shown in Figure 3.11. The raw

results are shown in Tables 3.2 to 3.7.The performance metrics used to evaluate the algorithms are precision, recall and F1 score, which were calculated respectively as $P = \frac{TP}{TP+FP}$, $R = \frac{TP}{N}$, $F = 2 \cdot \frac{P \cdot R}{P+R}$, where TP is the number of true positive detections, FP is the number of false positive detections and N is the number of test samples.

(a) No distractors

(b) Paper roll distractor

(c) 100PLUS can distractor

(d) Cup placed on its side

(e) Cluttered scene

(f) Robot arm occluding cup

Figure 3.11: Samples of cup detection test sets

The simplest test set has no distractor objects. A bar graph showing the performance metrics is shown in Figure 3.12. The precision of the image feature based methods (SIFT, SURF and ORB) is quite low, ranging from 0.1 to 0.35, as these algorithms are distracted by the background and give many false detections. Judging from the F1 score, which is the harmonic mean of the precision and recall, the CNN algorithm is the best performing algorithm here.

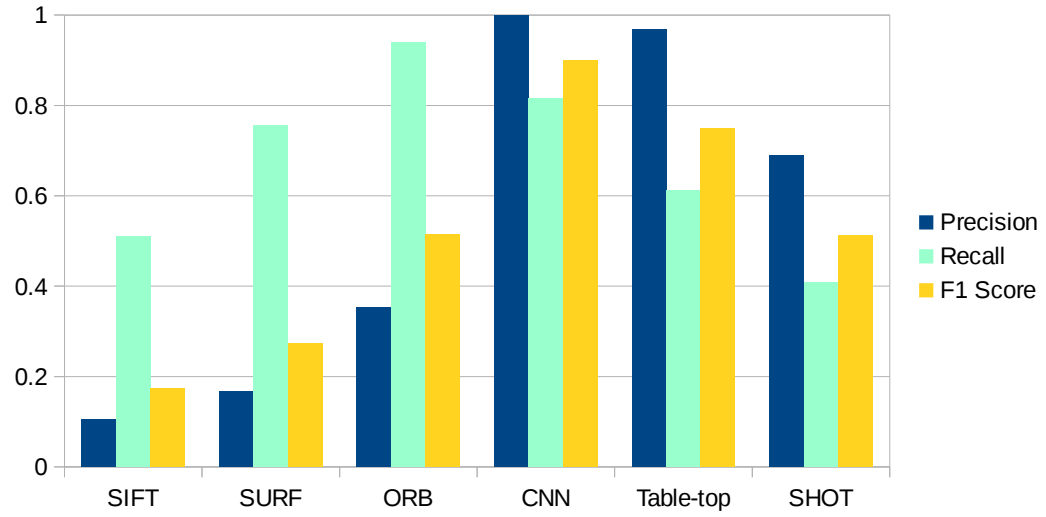


Figure 3.12: Precision, recall and F1 score graph for cup detection with no distractors

In the next test set, a single paper roll was used as a distractor, with the results shown in Figure 3.13. The paper roll is similar in shape and color to the cup, but is slightly larger. This seems to confuse both the CNN and point cloud algorithms (tabletop segmenter and SHOT), which have half the precision compared to the test case with no distractors. The CNN is still the best performing algorithm, and can likely be improved further by training it with more images of the cup with the paper roll.

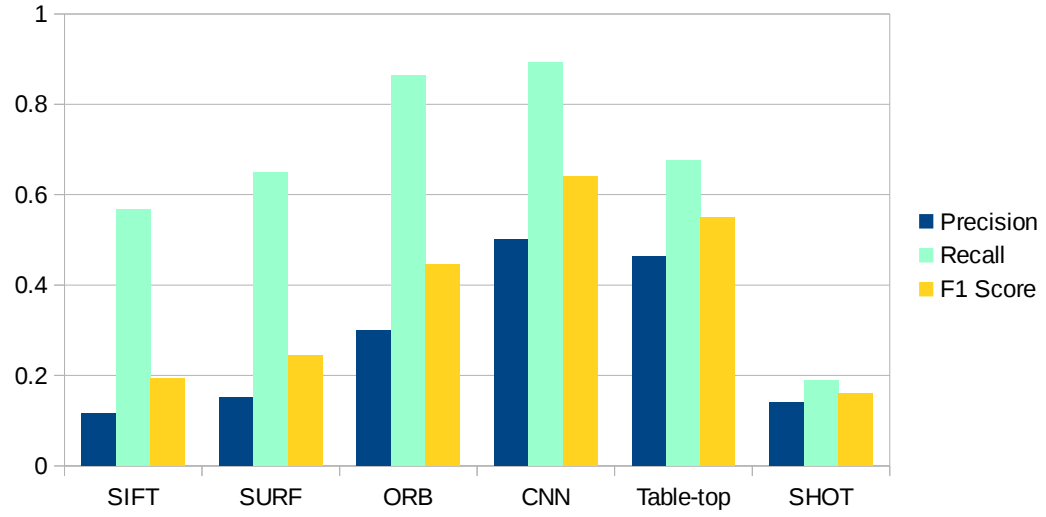


Figure 3.13: Precision, recall and F1 score graph for cup detection with paper roll distractor

A single 100PLUS soft drink can was also used as a distractor, with the results shown in Figure 3.14. The 100PLUS can is almost exactly the same size as the cup, but looks quite different. This time, the CNN has no trouble detecting the cup correctly, but the other algorithms still perform poorly.

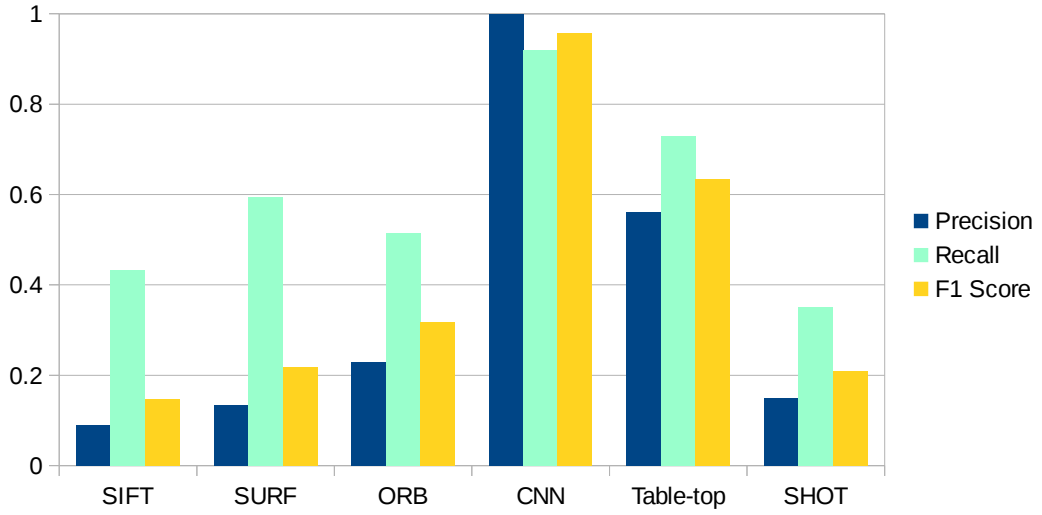


Figure 3.14: Precision, recall and F1 score graph for cup detection with 100PLUS distractor

In the next test set, the cup was placed on its side rather than upright. This can occur during the demonstration if the robot arm accidentally knocks the cup over. The results are shown in Figure 3.15. The point cloud algorithms seem to perform especially poorly in this case.

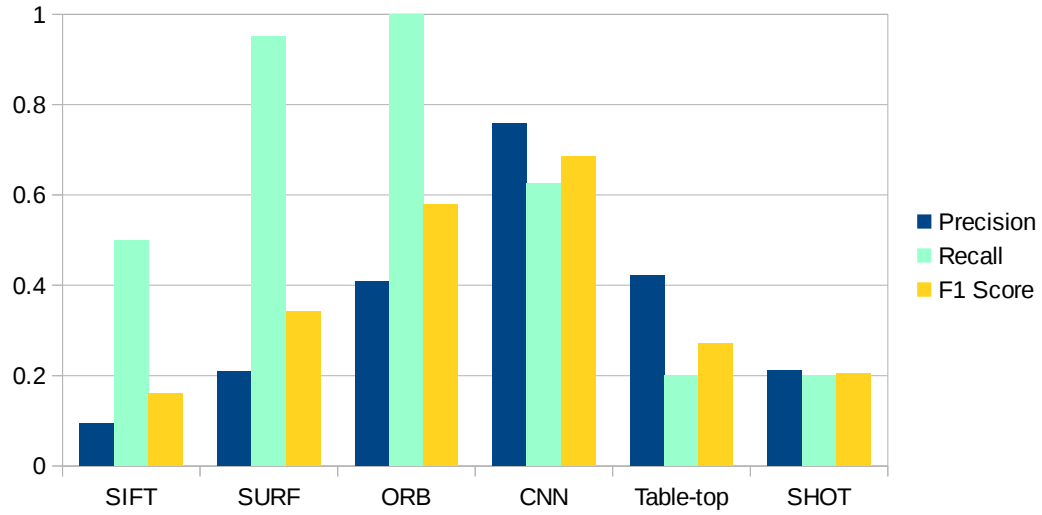


Figure 3.15: Precision, recall and F1 score graph for cup detection where the cup is placed on its side

Although the table will not be significantly cluttered during the conference demonstration, a test set with clutter was still produced to evaluate the algorithms. This is perhaps the most difficult test set, and this is reflected in the results shown in Figure 3.16. The feature based point cloud algorithm SHOT was unable to get a single correct detection. The CNN was the best performer, although it did get a fairly significant number of false detections.

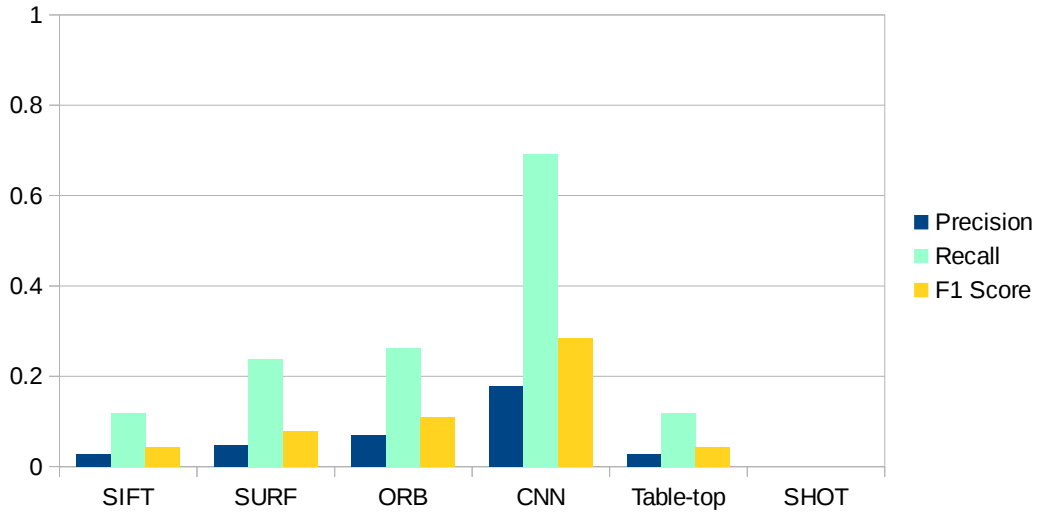


Figure 3.16: Precision, recall and F1 score graph for cup detection where the cup in a significantly cluttered scene

It may be helpful to the grasp planner if the cup's position can be tracked during the grasping procedure. In this case, the cup is partially occluded by the robot's arm or gripper. The results are shown in Figure 3.17. The point cloud algorithms (tabletop segmenter and SHOT) are the most significantly affected. If the gripper is touching the cup, the clustering algorithm of the tabletop segmenter cannot work well. Furthermore, the robot arm has many cylindrical components, which may be confusing to point cloud based algorithms.

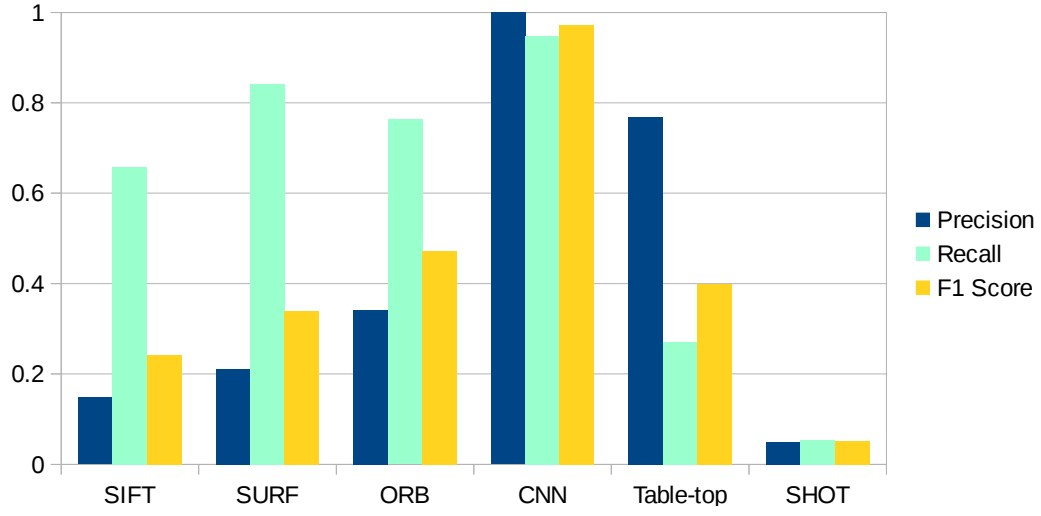


Figure 3.17: Precision, recall and F1 score graph for cup detection where the cup partially occluded by the robot arm

The cup used does not have many distinguishing features that make it easier to spot, except for a logo on its front. To investigate whether the cup is easier to detect with the logo, two test sets were collected, one with the logo visible to the camera, and one where the logo is not visible. The results are shown in Figure 3.18. The point cloud algorithms (table-top and SHOT), which do not process the RGB image, are mostly unaffected by the logo position. However, the image algorithms, particularly the CNN, show better performance when the logo is visible.

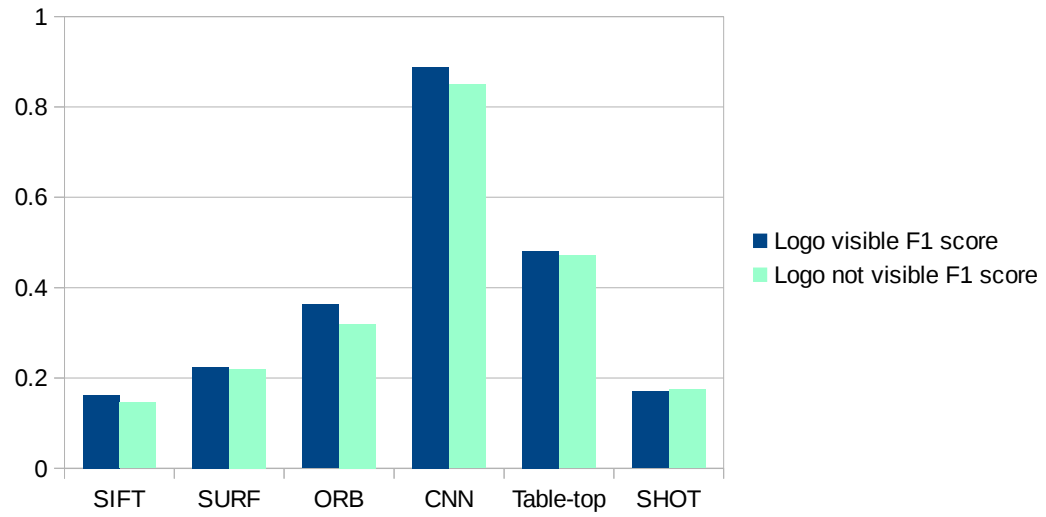


Figure 3.18: F1 scores comparing performance when logo is visible vs. not visible.

Test set	Number of samples	True positive detections	False positive detections
No distractors	49	40	0
Paper roll	37	33	33
100PLUS can	37	34	0
Cup placed on its side	40	25	8
Cluttered scene	42	37	8
Cup occluded by robot arm	38	36	0
Cup logo visible	91	88	19
Cup logo not visible	91	82	20

Table 3.2: Cup image detection with CNN

Test set	Number of images	True positive detections	False positive detections
No distractors	49	25	214
Paper roll	37	21	160
100PLUS can	37	16	163
Cup placed on its side	40	20	190
Cluttered scene	42	5	182
Cup occluded by robot arm	38	25	144
Cup logo visible	91	41	374
Cup logo not visible	91	38	394

Table 3.3: Cup image detection with SIFT

Test set	Number of images	True positive detections	False positive detections
No distractors	49	46	84
Paper roll	37	32	75
100PLUS can	37	19	64
Cup placed on its side	40	40	58
Cluttered scene	42	11	146
Cup occluded by robot arm	38	29	56
Cup logo visible	91	62	189
Cup logo not visible	91	56	204

Table 3.4: Cup image detection with SURF

Test set	Number of images	True positive detections	False positive detections
No distractors	49	46	84
Paper roll	37	32	75
100PLUS can	37	19	64
Cup placed on its side	40	40	58
Cluttered scene	42	11	146
Cup occluded by robot arm	38	29	56
Cup logo visible	103	74	208
Cup logo not visible	101	63	220

Table 3.5: Cup image detection with ORB

Test set	Number of images	True positive detections	False positive detections
No distractors	49	30	1
Paper roll	37	25	29
100PLUS can	37	27	21
Cup placed on its side	40	8	11
Cluttered scene	42	29	133
Cup occluded by robot arm	37	10	3
Cup logo visible	91	58	92
Cup logo not visible	91	57	94

Table 3.6: Cup point cloud detection with table-top algorithm

Test set	Number of images	True positive detections	False positive detections
No distractors	49	20	9
Paper roll	37	7	43
100PLUS can	37	13	74
Cup placed on its side	40	8	30
Cluttered scene	42	0	42
Cup occluded by robot arm	37	2	38
Cup logo visible	91	17	92
Cup logo not visible	91	19	108

Table 3.7: Cup point cloud detection with SHOT

3.3.2 Point Cloud Localization Accuracy

Sensor error is a major challenge in manipulation, and the Kinect is no exception. There seems to be conflicting reports from studies quantifying the Kinect's error. [68] measured an almost constant offset of 1.8 cm regardless of depth but [69] claims that the error is significantly dependent on the depth. Both papers agree that sunlight and varying sensor temperature can cause error. Sharp object curvature or highly reflective surfaces can also cause erroneous measurements or *flying pixels*. For example, note the *fuzziness* around the cup in Figure 3.20, which contributes to error in estimated object position.

An experiment was conducted to quantify the error in the object pose estimation. A cylindrical Pringles chip can was placed at varying positions on the table and the pose estimate from the Kinect recorded through the tabletop segmenter and pose tracker. The ground truth was measured through an OptiTrack motion capture system, which has sub-millimeter accuracy. The results are shown in Figure 3.19, where the x-axis is forward from the center of MOVO. The mean error is approximately 3.7 cm with a standard deviation of 0.9 cm, but as can be seen, the error depends on the distance and angle of the object relative to the Kinect sensor. The pose estimation error can be increasingly unpredictable if the object is too close to the Kinect. In our case, no calibrated offsets were hard-coded, and instead this significant uncertainty was handled through our POMDP motion planner.

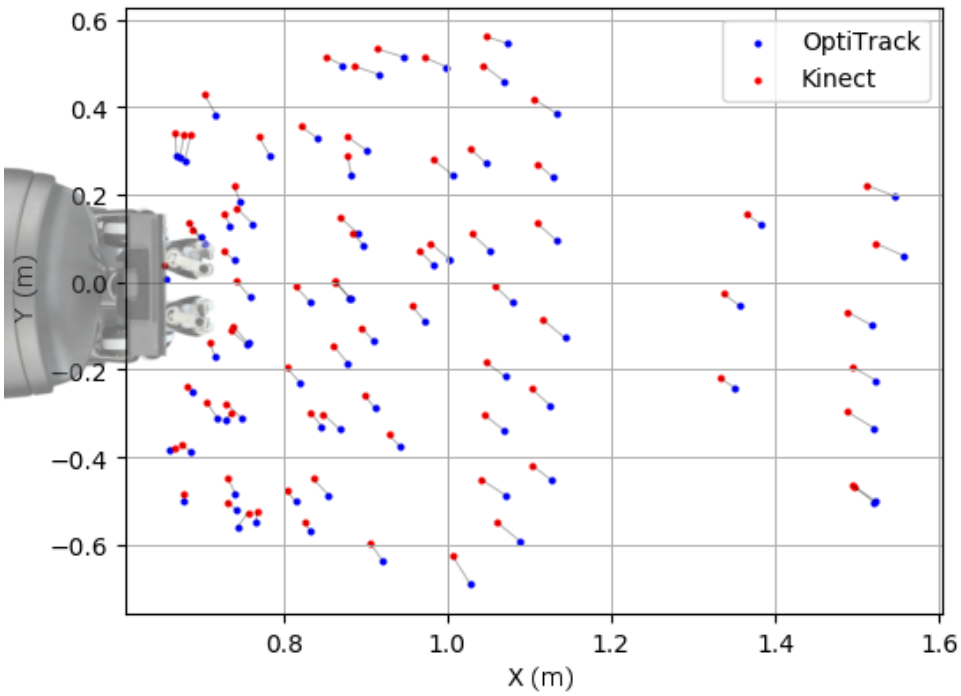


Figure 3.19: A top-down scatter plot comparing Kinect and OptiTrack pose estimates of an object. The MOVO robot is shown approximately to scale for reference.

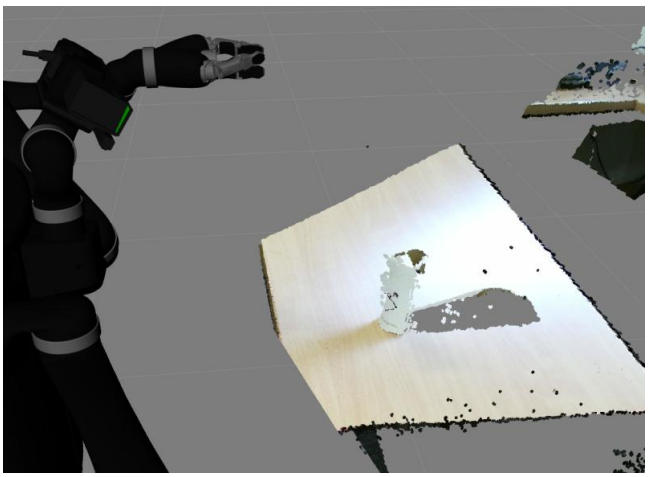


Figure 3.20: Point cloud of the cup on the table. The sharp curvature causes fuzziness around the cup.

3.4 Conclusion

The goal of this project was to develop a perception pipeline on the MOVO robot to enable a POMDP path planner to demonstrate its planning capabilities in a candy scooping demonstration. The final system consisted of a tabletop segmenter component, which segmented the table and obstacles, a pose tracker component, which can determine the cup's pose, and a CNN component, which provided reliable detection of the cup. Test results showed that the CNN has the best performance when compared to pure point cloud algorithms. The CNN also outperforms both image and point cloud feature-based algorithms, including SIFT, SURF, ORB and SHOT. The only downside to the CNN is that it is necessary to collect many training images, which can be time consuming. It was noted that detection accuracy may be improved if distinguishing features, such as a logo, are visible. Overall, the system performed well, succeeding in 98% of 150 runs completed in live demonstrations in the SIMPAR 2018 and ICRA 2018 conferences.

The following publication has been incorporated as Chapter 4.

[1] **Joshua Song**, Hanna Kurniawati, Exploiting trademark databases for robotic object fetching, *International Conference on Robotics and Automation (ICRA)*, 2019.

Contributor	Statement of contribution	%
Joshua Song	writing of text	70
	preparation of tables, figures	100
	theoretical derivations	50
	initial concept	40
	code development	100
	robot trials and experiments	60
Hanna Kurniawati	writing of text	30
	theoretical derivations	50
	initial concept	60
	robot trials and experiments	40
	supervision, guidance	100

Table 3.8: Contributions for ICRA paper

Chapter 4

Exploiting Trademark Databases for Robotic Object Fetching

4.1 Introduction

Object fetching, that is, getting a robot to recognize the object the user desires based on input from vision sensors and then retrieving it [18, 70], is an important task for service and assistive robots. These types of robots assist people at work or in their homes with daily tasks, which means to be effective, they must be able to recognize the large variety of objects in a household. However, such a recognition task remains an open problem. Convolutional Neural Networks (CNN) are advancing this domain tremendously, but require a huge amount of data. While category level (e.g. bottle, cup) classification is feasible, instance level (e.g. bottle of Pepsi or Coca-Cola, cup of Starbucks coffee) is more difficult since more data is required and the objects can be difficult to differentiate. There is work in progress on collecting 3D data for household objects, but this is a time consuming task and these datasets are currently incomplete [11, 12]. To alleviate the huge data requirement, we propose a novel approach: Exploit the availability of *structured data* from registered trademarks as a source of images and categorical information for both the CNN training and inference phase.

Companies place their logos on their products as distinguishing factors. Since companies would like to protect their logos, in general these logos are legally registered as trademarks. Organizations such as the World Intellectual Property Organization (WIPO) maintain databases of logos and their Nice Classification [71]. The Nice Classification identifies the goods and services category the logo was registered under. The list of goods is extensive and includes categories such as “soft drinks” and “soap”. Therefore, such databases contain a massive amount of labelled data, i.e., images of the logos labelled with their brand owners and product categories, which could be used for training a CNN.

However, the characteristics of images in trademark databases are very different from the images that a robot must work with: The logos images in the databases are not placed on products, the images do not contain any other logos nor objects, and are obviously not taken by a (relatively cheap) camera. Considering these differences, if used *as is*, the images in the trademark databases are unlikely to help

(a) Movo needs to find a soft drink in the cluttered shelf

(b) Detections are shown with trademark registration numbers and detection confidence. Trademark US-77585961 (Pepsi) is registered under soft drinks and thus satisfies the request.

Figure 4.1: Robot trial with logo detector trained only on synthetic RDSL images

with object recognition in robotics.

This paper presents an approach for object fetching in robotics to benefit from the massive amount of labelled data available in the trademark databases, despite the aforementioned difficulties. In particular, we propose two uses of the trademark datasets. First is for object recognition. We propose an extension of domain randomization, called Randomization-based Data Synthesizer for Logos (RDSL), that converts the trademark images of logos into synthetic "camera images". To this end, RDSL starts by automatically downloading the logo images from trademark databases and converting them into simple 3D scenes of logos (i.e., logos placed on simple geometrical objects). It then

randomizes various parameters related to the logo placement, object placement, camera parameters, and background environment. The entire process of RDSL is automated. The synthetic images it generates are used as training data for a CNN for the object recognition component of an object fetching task.

The second proposed use of trademark databases is a simple method that exploits the categorical information to help the robot in understanding users' request. In particular, it will allow users to ask the robot to retrieve objects based on both categorical information (e.g., soft drinks or soap) or brands (e.g., Coke or Pepsi).

We test RDSL on a benchmark problem for logo detection (i.e., FlickrLogos-32 [72]) and test the entire object fetching pipeline on a Kinova Movo robot. The results are promising. The logo detectors that were trained using only synthetic data generated by RDSL outperformed previous logo detectors trained on synthetic data. We also demonstrate that the logo detector trained using only data generated by RDSL can be successfully used for object fetching tasks.

4.2 Related Work

Early approaches to logo detection used keypoint detectors, such as SIFT [55, 56]. More recent work has shown that CNNs outperform previous methods for logo recognition [57–59]. The improvements in computer vision have been employed by several companies in order to provide query-by-image services for logos. For example, TrademarkVision [60] allows a user to upload a logo image in order to find similar logos for the purpose of finding copyright infringement.

The CNN architectures referenced in this paper include VGG [73] and MobileNet [43]. MobileNet is a smaller and faster network while still having similar accuracy to VGG.

In a classification configuration, CNNs are invariant with regard to the position of the object within the image. However, it is often useful to localize and identify multiple objects within the image; this is termed object detection. One approach to this is Regions with CNN features (R-CNN) [46], which first finds regions of interest based on color and texture, then classifies the regions through a CNN and Support Vector Machine (SVM). Detection speed was improved with Fast R-CNN [47] and Faster R-CNN [10]. Faster R-CNN uses convolution features for both region proposals and class prediction. These convolution features may be taken from classification CNNs. Example configurations include Faster R-CNN with ResNet, Faster R-CNN with Inception and so on. An even faster object detection method (though slightly less accurate) is Single Shot MultiBox Detector (SSD) [9], which eliminates proposal generation and performs all computation in a single network.

In addition to class labels, object detectors also require their training data to be annotated with either bounding boxes or pixel masks for each object instance. Publicly available datasets include COCO [50] and Open Images [51], which are datasets for 91 and 600 object categories, respectively. Ammirato et al. [52] modified general object detectors trained on such datasets to be able to detect specific object instances (e.g. my mug instead of any mug), given several new images of the object instance.

Instead of training a new CNN model from scratch, the time and data required for training can be reduced through a process called transfer learning or fine-tuning [44]. In this process, a model that has been trained for a certain task is re-trained on data for the new task.

The training data can be expanded through augmentation techniques such as rotating, flipping and cropping images [53]. In addition to data augmentation, new training data can be generated through synthetic means. Eggert et al. [58] and Su et al. [59] generated synthetic data for logos by applying random geometric and illumination transforms to logos and pasting them on random photos. Montserrat et al. [62] followed a similar approach but used a CNN to estimate depth in the background image in order to blend the logo image more realistically.

Recent work [14] used simulators and domain randomization to generate synthetic data. The parameters of the simulator are randomized in unrealistic ways in order to force the CNN to learn the essential features of the target object. The synthetic data was then used to train an object detector for robot localization and grasping. They suggest that "with enough variability in the simulator, the real world may appear to the model as just another variation" [14]. Tremblay et al. [13] introduced *flying distractors* as a domain randomization component, which are random geometric shapes added to the scene. They then demonstrated the effectiveness of domain randomization for car detection.

Hinterstoisser et al. [54] generated synthetic data for a household objects detector by composing CAD models of those objects on random background images. They found that freezing the feature extractor layers and only fine-tuning the region proposal layers improved performance.

Similar to the above methods, our synthetic data generator, RDSL, also uses randomization. However, unlike the above methods that start with a relatively meaningful simulated scene (e.g., a table for [14] and road for [13]), in our problem, we do not have any such initial scene. In fact, the input images we have for the data generator are stand-alone logos, while in the real world, these logos are placed on various different objects.

4.3 Using Trademark Database for Fetching

Our pipeline for fetching tasks consists of two phases, a training phase and an execution phase. In the training phase, logo images are downloaded from the database and processed through our synthetic data generator (RDSL) described in Section 4.3.1. The synthetic images are then used to train a Convolutional Neural Network (CNN) object detector. In the execution phase described in Section 4.3.2, the robot uses the CNN to identify the object requested by the human user.

4.3.1 RDSL: A Method for Synthetic Data Generation

RDSL uses the logos from trademark databases to generate synthetic images that can be used to train a CNN logo detector for use in robotics applications. The entire process of this synthetic data generation is automated, therefore new logos can be generated with minimal human intervention.

RDSL starts by downloading the required logo images from trademark databases. These images are then preprocessed by removing the image's background, which for these images can be done easily using existing image editing tools, such as ImageMagick. The logo images were then superimposed on shapes such as cylinders, boxes and planes. The "UV maps" are specified for each shape to ensure that the 2D texture is correctly mapped in such a way that the logo is in the front of the object and hence visible to the camera.

To increase variability in the synthetic images, so as to improve the classifier's ability to generalize to real images, RDSL randomizes the following parameters:

- The number of objects of interest and distractor objects.
- Each object's properties: Color, texture, shape, materials (e.g., roughness and metalness), position, orientation, dimensions, and position of the logo on the object.
- Camera position and orientation.
- Lighting properties: Number, color, and brightness.
- Floor and wall materials.
- Additional noise, blurring, and coloring.

Furthermore, RDSL applies rendering techniques such as bump mapping, which simulates bumps and wrinkles during lighting calculations. For example a tile bump map was used to generate Figure 4.2a. A few other examples of the synthetic images generated are shown in Figures 4.2c-f.

In addition to the logo class label, the training data also needs to be labeled with the pixel coordinates of a bounding box around the logo. This was done by performing an additional render of the logo alone as a separate mask image as shown in Figure 4.2b. It is then trivial to extract the bounding box from the mask image.

A summary of the steps followed to generate the synthetic images is given in Pseudocode 2. RDSL can use any available 3D modelling tools, such as Autodesk 3ds Max, Unreal Engine and Blender. It can also use any renderer.

4.3.2 Identifying Objects Through Logos

Our method is summarized in Figure 4.3. During the execution phase, the human user can either request for a particular brand, such as Pepsi or Pringles, or can make a request for a Nice Classification (NCL). The Nice Classification classifies the goods and services the logo is used for. Examples of Nice classes include soft drinks or potato chips. If the user made a brand request, the brand's Nice Classification can be looked up from the database. For the small number of logos used in this trial, the database can be stored onboard rather than needing to make a query over the Internet.

In this work, we assume that the user will not provide any directions on where the target object is located. Instead, given a map of the environment and the target object's Nice Classification, the robot

(a) Logo with tile bump map

(b) Image mask for (a)

(c) Logo with distractors

(d) ART renderered image

(e) Logos on bottle shapes

(f) Noise and color shift

Figure 4.2: Synthetic logo images generated through RDSL

can narrow down the possible locations the object may be. For example, a household map can specify that food and drinks are located in the fridge, soap on a certain shelf, and so on.

The robot can then navigate to each location and perform a scan by panning the camera over the scene and passing the RGB frames to the object detection CNN. The CNN provides a bounding box of the logo in pixel coordinates, which must then be converted to world coordinates (i.e. XYZ coordinates in meters) in order to plan the grasping motion. This conversion can be done by using a distance measurement and the intrinsic and extrinsic matrices of the camera.

If the user made a brand name request, the exact matching logo must be returned. However, if the user made a Nice Classification request, then any logo under that classification can be returned.

Pseudocode 2 Synthetic data generation

```
function GENERATE MATERIAL
    Create a material from a random RGB value OR select a random texture
    Randomize material properties such as roughness and metalness
    Occasionally apply a bump map
return material
end function
for each selected logo do
    Download logo
    Remove logo's background
end for
function RDSL
    for the number of synthetic images required do
        for 1 to 3 randomly selected logos do
            Randomize logo size while maintaining aspect ratio
            m ← GENERATE MATERIAL
            Composite logo on m
            Place m on a random shape
            Randomize the shape's position, orientation and dimensions
        end for
        Randomize the camera position and have it point towards the logos
        for a random number of distractor objects do
            m2 ← GENERATE MATERIAL
            Place m2 on a random shape
            Randomize the shape's position, orientation and dimensions while ensuring it does not
            block line of sight between camera and logo
        end for
        Randomize the number of lights and their color and brightness
        Randomize floor and wall materials
        Randomize noise, blurring, coloring
        Render image to file
        Render image mask to file
    end for
end function
```

4.4 Experimental Results

The purpose of our experiments are two-fold. First, we compare the performance of the synthetic data generation method RDSL to existing synthetic data generation methods on benchmark test sets. Second, we test the applicability of our fetching pipeline (Section 4.3) on a robotic platform and provide qualitative results.

4.4.1 Benchmark Tests

In this test, we used RDSL to generate 100 synthetic images for each logo in the FlickrLogos-32 data set. The logos were downloaded from TrademarkVision's database through their API. The randomization procedure of RDSL is implemented as a script inside Autodesk 3ds Max 2018. We limit the type of objects (both objects of interest and distractor objects in Pseudocode 2) to simple geometry,

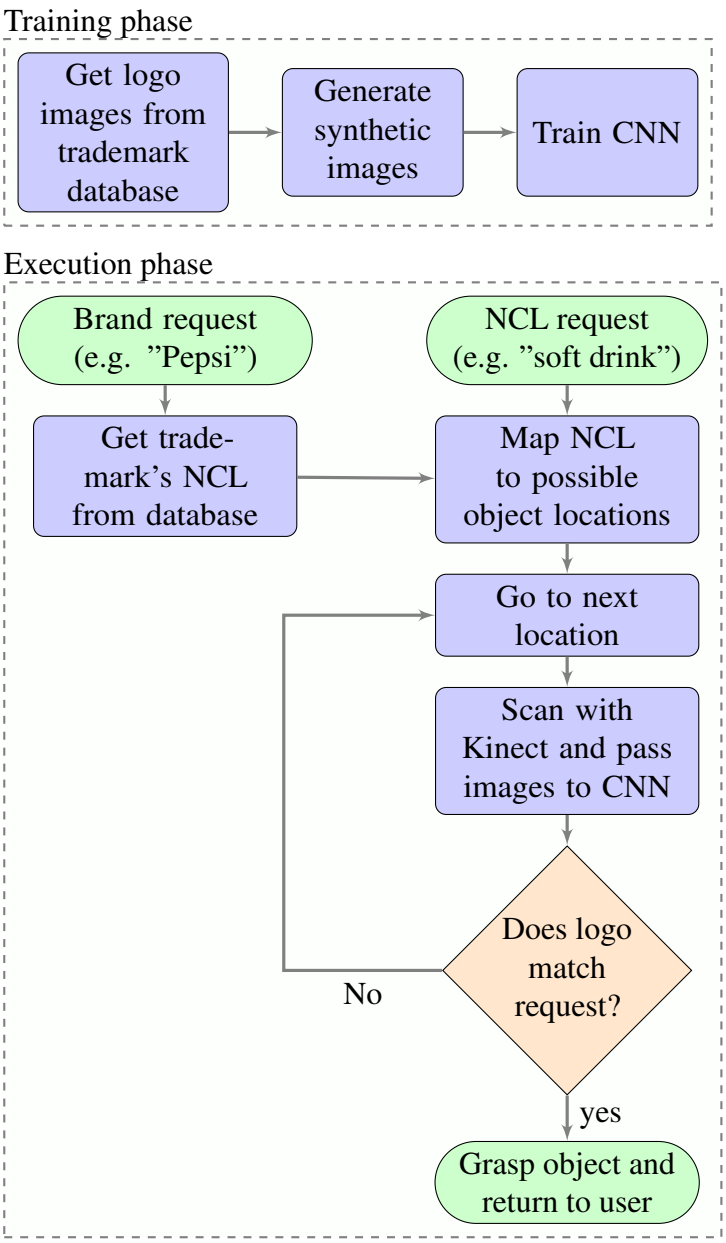


Figure 4.3: Procedure for robotic object fetching with logos

such as cylinders, boxes, etc. and only use textures from the Autodesk libraries for generating the synthetic data. The scanline renderer was the primary rendering engine used. While the Autodesk Raytracer (ART) renderer is able to produce more realistic images and is compatible with a wider range of 3ds Max materials, it is significantly slower to render and did not produce a noticeable effect on the classifier performance.

The CNNs were trained using the Tensorflow [63] framework. Two different object detection architectures were used, SSD with MobileNet and Faster RCNN with VGG16. SSD-MobileNet was trained for use in our robotics task where recognition needs to be carried out quickly and on a computationally limited platform. Faster RCNN-VGG16 was trained for benchmarking against previous work. The models were pre-trained on COCO and then fine-tuned for 200k steps on either the real photos from the FlickrLogos data set or the synthetic data generated through the process described in Section 4.3.1. For comparison purposes, we also list the results as reported in [59, 62].

Training and benchmarking was done on a NVIDIA P100 GPU. The FlickrLogos [72] test set was

Title	Network Architecture	Training images per logo	Adidas Corona Google Ritt	Aldi DHL Guin Shell	Apple Erdi Hein Sing	Becks Esso HP Starb	BMW Fedex Milka Stel	Carls Ferra Nvid Texa	Chim Ford Paul Tsin	Coke Fost Pepsi Ups	mAP
RDSL (fast)	SSD MobileNet	**100 Syn**	44.4 81.5 65.8 54.2	49.4 27.6 7.0 27.3	19.5 40.7 68.5 40.9	62.0 64.3 21.5 90.1	80.8 48.2 11.9 83.6	49.0 72.9 1.0 75.9	43.3 79.3 57.6 72.9	22.1 51.7 19.2 22.8	**48.6**
Real (fast)	SSD MobileNet	40 Real	30.6 92.7 77.6 58.4	45.3 56.1 85.0 37.0	65.8 40.1 61.9 69.8	60.0 88.7 24.3 91.1	72.1 63.4 36.1 41.3	47.9 83.6 31.4 73.7	61.8 79.2 84.5 72.2	37.5 72.3 29.6 59.5	60.3
Real + RDSL Synthetic (fast)	SSD MobileNet	40 Real + 100 Syn	61.4 99.3 79.9 76.9	70.5 53.8 85.6 37.1	77.5 59.5 78.3 95.3	69.9 92.7 53.3 97.0	85.9 76.6 48.0 85.2	66.4 86.7 50.4 89.4	73.8 87.3 95.5 81.7	70.8 86.3 47.1 71.8	74.7
RDSL	Faster RCNN VGG16	100 Syn	38.8 18.9 47.0 50.5	55.3 37.5 4.6 31.9	35.8 55.5 50.8 33.4	18.8 65.7 46.6 71.6	59.4 40.6 9.5 80.2	15.5 64.0 14.4 53.1	28.7 35.3 74.3 69.3	26.8 79.5 21.4 39.6	42.9
Real	Faster RCNN VGG16	40 Real	52.3 95.9 93.5 77.9	81.6 82.0 93.2 52.7	77.6 96.3 75.7 91.1	69.9 90.1 51.3 99.4	79.4 75.5 56.0 81.7	66.3 90.2 51.9 85.3	81.8 83.8 97.6 85.1	71.0 88.7 54.4 84.8	78.6
Real + RDSL Synthetic	Faster RCNN VGG16	**40 Real + 100 Syn**	64.8 96.9 95.6 80.5	82.1 82.9 96.2 53.7	87.4 90.4 72.6 95.2	80.8 86.9 63.9 99.0	84.6 77.5 50.4 92.9	72.3 88.6 58.9 89.6	82.8 87.2 98.7 87.4	76.9 90.6 65.9 89.6	**82.0**
RealImg [59]	Faster RCNN	40 Real	68.1 90.9 98.0 81.0	79.1 77.4 90.7 57.3	84.5 90.9 81.3 97.9	72.3 88.6 67.0 99.5	86.4 71.1 54.5 86.7	68.0 91.0 64.0 90.4	78.0 98.3 90.9 87.5	73.3 86.2 59.6 85.8	81.1
SynImg-32Cls [59]	Faster RCNN	100 Syn	9.4 6.1 23.0 22.7	47.3 11.1 16.7 38.3	9.6 4.1 43.1 15.5	70.3 44.7 9.9 65.6	39.9 22.9 4.6 28.7	28.3 60.9 1.1 55.1	15.8 43.6 38.1 27.4	21.7 28.8 9.7 20.1	27.6
SynthLogo [62]	Faster RCNN VGG16	≈ 460 Syn	Not reported								47.7

Table 4.1: FlickrLogos-32 test set (3,960 images) average precision benchmark results. The first six rows are results from our runs. The last three rows are results from the respective papers.

used for benchmarking. It consists of 3,960 real world photos of the logos in various settings. As expected, SSD-MobileNet is faster, taking 198 seconds to completely classify the test set, compared to 918 seconds required by Faster RCNN-VGG16. Table 4.1 shows average precision values for each logo as well as the Mean Average Precision (mAP).

SSD-MobileNet trained on RDSL generated synthetic data outperforms previous work on logo detectors trained on synthetic data (SynImg-32Cls and SynthLogo). Moreover, mixing real and our synthetic data resulted in higher performance than using real data alone. These results are in-line with the results from Su et al. [59] who found that mixing synthetic data and real data can improve performance if there is a shortage of real data.

When trained on real images, Faster RCNN-VGG16 was more precise than SSD-MobileNet by 5.2%. Surprisingly, Faster RCNN-VGG16 performs worse by 11% when trained on synthetic data. A possible explanation is that the smaller SSD-MobileNet network does not overfit to our synthetic data. We followed the method used in [54] for avoiding overfitting to synthetic data and first tried freezing

the entire VGG16 feature extractor network, then freezing only the first convolution layer, but neither method improved precision.

Transfer learning (or domain adaptation) techniques, such as in [74, 75] have previously been shown to improve the accuracy of CNNs trained on synthetic images, but were not implemented due to two main considerations: we wished to compare our synthetic data (and not network architectures) to previous work, which did not employ these techniques, and test how well our simple strategy of using 0 real image for training applies to situations where images are noisy.

The precision appears to depends heavily on the logo. For SSD-MobileNet trained on our synthetic data, the Starbucks logo has an average precision of 90.1 compared to 19.2 for the Pepsi logo. This is also apparent for the detectors trained on real images. It is possible that the set of test images for certain logos are more difficult than others, or perhaps some logos are simply more recognizable to the detector.

We performed an ablation study where we disabled certain randomization features one at a time while keeping the other features constant and measured the effect on the classifier accuracy. Removing the random shapes for the logo resulted in a mean average precision decrease of 8.4, not having random background textures caused a decrease by 9.6, and not having bump maps caused a decrease by 1.2. The FlickrLogos-32 dataset contains many logos on bottles, therefore we experimented with having bottle shapes in our synthetic data, however this did not improve performance.

4.4.2 Robot Trials

Movo [21] was used for performing a real-world object fetching trial. Movo is a mobile robotic platform equipped with two 6 DOF (degrees of freedom) arms, a Kinect and a 2D planar laser sensor. The Kinect for Xbox One is a time-of-flight sensor that provides both RGB images and distance measurement. Movo is also equipped with two Intel NUC5I7RYH computers, and comes ready to use with Robot Operating System (ROS) [22] and its motion planning framework MoveIt! [27].

To perform a scan for logos, Movo pans the Kinect over the scene and passes the RGB frames to the object detection CNN, which processes frames at a rate of 3 Hz. We trained SSD-MobileNet for the logos relevant to our scenario, which are trademarks that may appear in a typical household. The CNN was trained on 200 synthetic images per logo. We used an input image size of 600x600 instead of the default 300x300 as we found this performed better for small logos. The CNN provides a bounding box of the logo in pixel coordinates, which must then be converted to world coordinates. This conversion can be done by using the distance measurement and the intrinsic and extrinsic matrices. ROS's Kinect interface already performs this calculation in order to convert the depth image from the Kinect into a point cloud, which is a set of XYZ points. Therefore, we found it more convenient to extract the world coordinates by calculating the pixel's array index in the point cloud.

A scenario is shown in Figure 4.4. The products used here were soap (Dettol), chewing gum (Extra), potato crisps (Pringles), and soft drinks (Sprite, A&W, 100Plus). In this case, the user has asked Movo to bring food. We have placed the objects in such a way that the logos are facing Movo. If

the logos were not visible, the objects can be detected by finding clusters in the point cloud and then rotated with the robot arm. After navigating to the table, Movo performed a scan. The detections from the logo detector are shown in Figure 4.4a with the trademark registration numbers. Unfortunately, the logo detector does struggle with small logos, in this case the Dettol logo. For each logo detected, Movo looked up the Nice Classification and then grasped Pringles, which is registered under class 29 (foodstuffs of animal or vegetable origin), and is also registered under "Snack foods, potato chips and potato crisps".

Our logo detector also works in visually cluttered scenes such as the one shown in Figure 4.1. Such scenes are often problematic for traditional, point cloud based object detection methods. This and other scenarios may be viewed in the accompanying video.

An issue that may cause confusion to the robot is that certain logos may be registered under certain Nice Classification while not having products for those classes. For example, the Mitsubishi logo is not only registered under vehicles, but also foods and drinks. We considered cross-checking the trademark database with Open Product Data [76], which is an open-source database of products, but found that database is currently quite incomplete.

4.5 Conclusion

This paper investigated the use of trademarks to improve object recognition for fetching tasks by service robots. Rather than manually collecting training data on objects a robot may encounter, which is tedious and time-consuming, we leverage existing structured data from trademark databases. To this end, we propose an automatic data synthesizer, RDSL, that takes trademark images as its only input and generates synthetic labelled data for training a CNN-based logo detector. Furthermore, during execution, we leverage categorical information (i.e., Nice Classification) from the trademark databases to help the robot identify the object that the human user requested. We demonstrated the effectiveness of our approach via experiments on benchmark logo detection problems and a fetching task on the Kinova Movo mobile manipulator.

There is still much room for future work, including improving the quality of synthetic data generated, implementing transfer learning techniques to resolve the domain shift, and understanding the distribution for randomization that will be most suitable for robotics tasks. In this paper, all randomization used a uniform distribution. However, it is likely that other distributions may cover the span of real-world scenarios better. We hope the ideas and results presented here will encourage the exploration of other ways of alleviating a common issue in robotics: the need for a huge amount of relevant, labeled data.

(a) Movo needs to identify which object is food

(b) Movo grasps Pringles, which is trademarked under food

Figure 4.4: Trial with logo detector trained on RDSL images

Chapter 5

Summary

This thesis was focused on the problem of object recognition in the context of service robots operating in households. A MOVO robot was used for experiments, and so in this thesis a perception pipeline was developed for MOVO. A candy scooping task was used to demonstrate a POMDP motion planner in the SIMPAR 2018 and ICRA 2018 conferences. In order to detect the cup used as a scoop, a range of object detection algorithms were tested, and CNNs were found to be the most reliable. Overall, this system performed well, succeeding in 98% of 150 runs during the live demonstrations. However, CNNs require many training images of the object, which can be tedious to collect.

To alleviate this issue, the novel idea of using information from trademark databases was explored. The logo images from the databases were used for training an object detector, and the logo's Nice classification was used as a source of semantic information about the objects. In order to translate the logo images into a form more suitable for training CNNs, RDSL (Randomization-based Data Synthesizer) was developed based on ideas from domain randomization. RDSL uses 3D rendering software to automatically generate synthetic data from the databases logo images. A CNN logo detector trained on RDSL synthetic data outperformed previous logo detectors trained on synthetic data. The use of this logo detector was also demonstrated in a practical implementation for object fetching by MOVO. Tests on this robot indicated promising results, despite not using any manually-labelled real world photos for training

An issue encountered was that small logos may be difficult to detect. A possible solution could be to mount a camera on the robot's hand, allowing it to closely inspect objects. Another issue was that some logos may have "noisy" Nice Classifications, but this could be solved in the future by cross-checking with product databases. Future work could also incorporate barcode lookup and optical character recognition in order to read brand names. A service robot would likely benefit from having multiple types of object recognition methods at its disposal.

Overall, it can be concluded that utilizing data from trademark databases or other similar sources is a promising avenue for providing service robots with the ability to recognize household objects.

Bibliography

[1] J. Song and H. Kurniawati. Exploiting trademark databases for robotic object fetching. In *International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 20-25 2019.

[2] M. Hoerger, J. Song, H. Kurniawati, and A. Elfes. Pomdp-based candy server: Lessons learned from a seven day demo. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Berkeley, USA, July 11-15 2019.

[3] A. J. Snoswell, V. Dewanto, M. Hoerger, J. Song, H. Kurniawati, and S. P. N. Singh. A distributed, any-time robot architecture for robust manipulation. In *Australasian Conference on Robotics and Automation (ACRA)*, Canterbury, New Zealand, Dec 4-6 2018.

[4] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *European conference on computer vision*, pages 356–369. Springer, 2010.

[5] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, 54(10):95–103, 2011.

[6] J. Redmon. Yolo: Real-time object detection, 2016.

[7] S. Niekum and I. I. Saito. Ar track alvar, 2018.

[8] UC San Diego. A roadmap for u.s. robotics: From internet to robotics, 2016.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[10] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[11] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 509–516. IEEE, 2014.

[12] A. Kasper, Z. Xue, and R. Dillmann. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012.

[13] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *arXiv preprint arXiv:1804.06516*, 2018.

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[15] IFR. Service robots, 2018.

[16] IFR. Wr 2018 service robots executive summary, 2018.

[17] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5, 2010.

[18] M. Mast, M. Burmester, B. Graf, F. Weisshardt, G. Arbeiter, M. Španěl, Z. Materna, P. Smrž, and G. Kronreif. Design of the human-robot interaction for a semi-autonomous service robot to assist elderly people. In *Ambient Assisted Living*, pages 15–29. Springer, 2015.

[19] J. Fischer, G. Arbeiter, R. Bormann, and A. Verl. A framework for object training and 6 dof pose estimation. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–6. VDE, 2012.

[20] Willow Garage. Pr2, 2018.

[21] Kinova. Movo mobile manipulator, 2018.

[22] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[23] P. Svestka, J. Latombe, and L. Overmars Kavraki. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[25] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[26] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[27] I. A. Sucan and S. Chitta. Moveit!, 2018.

[28] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[29] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[30] A. Somani, N. Ye, D. Hsu, and W. S. Lee. Despot: Online pomdp planning with regularization. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1772–1780. Curran Associates, Inc., 2013.

[31] H. Kurniawati and V. Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer, 2016.

[32] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[33] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[34] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[35] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[36] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010.

[37] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, 2010.

[38] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

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

[40] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

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

[42] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[44] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

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

[46] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[47] R. Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[48] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[49] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[50] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[51] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[52] P. Ammirato, C.-Y. Fu, M. Shvets, J. Kosecka, and A. C. Berg. Target driven instance detection. *arXiv preprint arXiv:1803.04610*, 2018.

[53] J. Wang and L. Perez. The effectiveness of data augmentation in image classification using deep learning. Technical report, Technical report, 2017.

[54] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige. On pre-trained image features and synthetic images for deep learning. *arXiv preprint arXiv:1710.10710*, 2017.

[55] A. D. Bagdanov, L. Ballan, M. Bertini, and A. Del Bimbo. Trademark matching and retrieval in sports video databases. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 79–86. ACM, 2007.

[56] J. Kleban, X. Xie, and W.-Y. Ma. Spatial pyramid mining for logo detection in natural scenes. In *Multimedia and Expo, 2008 IEEE International Conference on*, pages 1077–1080. IEEE, 2008.

[57] S. Bianco, M. Buzzelli, D. Mazzini, and R. Schettini. Deep learning for logo recognition. *Neurocomputing*, 245:23–30, 2017.

[58] C. Eggert, A. Winschel, and R. Lienhart. On the benefit of synthetic data for company logo detection. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1283–1286. ACM, 2015.

[59] H. Su, X. Zhu, and S. Gong. Deep learning logo detection with data expansion by synthesising context. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 530–539. IEEE, 2017.

[60] TrademarkVision. Trademarkvision, 2018.

[61] LogoGrab. Logograb, 2018.

[62] D. M. Montserrat, Q. Lin, J. Allebach, and E. J. Delp. Logo detection and recognition with synthetic images. *Electronic Imaging*, 2018(10):337–1, 2018.

[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[64] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520. IEEE, 2018.

[65] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[66] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[67] G. Bradski and A. Kaehler. Opencv. *Dr. Dobbs journal of software tools*, 3, 2000.

[68] O. Wasenmüller and D. Stricker. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Asian Conference on Computer Vision*, pages 34–45. Springer, 2016.

[69] T. Breuer, C. Bodensteiner, and M. Arens. Low-cost commodity depth sensor comparison and accuracy analysis. In *Electro-Optical Remote Sensing, Photonic Technologies, and Applications VIII; and Military Applications in Hyperspectral Imaging and High Spatial Resolution Sensing II*, volume 9250, page 92500G. International Society for Optics and Photonics, 2014.

[70] H. Huttenrauch and K. S. Eklundh. Fetch-and-carry with cero: observations from a long-term user study with a service robot. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 158–163. IEEE, 2002.

[71] WIPO. Nice classification, 2018.

[72] S. Romberg, L. G. Pueyo, R. Lienhart, and R. van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 25:1–25:8, New York, NY, USA, 2011. ACM.

[73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[74] Z. Murez, S. Kolouri, D. Kriegman, R. Ramamoorthi, and K. Kim. Image to image translation for domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4500–4509, 2018.

[75] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. Van Gool. Domain adaptive faster r-cnn for object detection in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3339–3348, 2018.

[76] Open Knowledge Labs. Open product data, 2018.