Merging Machine Learning and 3D Imaging Methods for Complex Robot-Object Interaction Systems

Adrián Llopart Maurín

Ph.D Thesis

2018

DTU Electrical Engineering
Department of Electrical Engineering

English title of the thesis:
Merging Machine Learning and 3D Imaging Methods for Complex Robot-Object Interaction Systems

Afhandlingens danske titel:
Fusion af maskinlæring og 3D-billedbehandlingsmetoder i komplekse robot-objekt interaktionssystemer

PhD Student:
Adrián Llopart Maurín
adllo@elektro.dtu.dk
ORCID: 0000-0003-0900-3446

Supervisors:
Ole Ravn
or@elektro.dtu.dk
ORCID: 0000-0003-2265-0031

Niels A. Andersen
naa@dtu.dk
ORCID: 0000-0002-9985-6444

Jong-Hwan Kim
johkim@rit.kaist.ac.kr

Technical University of Denmark
Department of Electrical Engineering
Section of Automation and Control
Elektrovej, Building 326
DK-2800 Kgs. Lyngby
Denmark
Phone: (+45) 45 25 34 76
Email: info@elektro.dtu.dk
www.elektro.dtu.dk

Summary (English)

With the constant biomedical and technological advances, the worlds economic and social structures will be challenged in the following years: on the one side, a large amount of jobs will become automated and the work paradigm will shift from low paying repetitive tasks to high paying technical jobs. On the other side, the population pyramids will invert and so the amount of dependable people will be equal to those in the working age. These disruptive transformations will, without a doubt, require the development of robots capable of understanding and interacting with their surroundings; either to carry out mundane and risky tasks or to take care of the household, the elderly or the children. Therefore, the development of service, intelligent robots, is becoming more a necessity than a luxury. These robots must be able move in, perceive and interact with the world around them.

The main goal of this research has been the development of a general perception pipeline for robotic platforms to recognize, detect and segment objects in a dynamic and cluttered environment, with no prior knowledge. To this end, the synergy between Machine Learning (Deep Neural Networks) and 3D segmentation methods was explored. The Deep Learning foundations, architectures and mathematical theory have been assessed to find the best suited network models for solving the object recognition, detection and segmentation tasks. Two main state-of-the-art network architectures were evaluated, retrained and implemented for diverse applications: YOLO and Mask-RCNN. The datasets employed were both standard (COCO, SUNRGB-D, ImageNet) and custom (Doors and cabinets, Maritime). The 2D instance segmentation derived from the forward pass of the networks works as a Region of Interest that masks a rectified depth image. 3D points contained within the objects physical space were then generated. Noise and background information was filtered using several geometrical segmentation procedures based on the objects label; these include primitive fitting RANSAC, euclidean distance and region growing segmentation. Occlusions and geometrical errors when reconstructing the objects model were reduced through multi-viewpoint perspectives and point cloud registration using the ICP/NICP algorithms. Thus, 3D models of all recognized classes were derived simultaneously, in real time, with a high degree of precision and robustness. From said reconstructions, additional information can be obtained such as position, orientation, size and grasping data. The robots pre-learnt behaviours were adapted based on the data extracted from the surroundings; otherwise, new behaviours were inferred from previous knowledge.

This perception pipeline was also integrated into more complex systems (from small mobile to humanoid robots) to demonstrate the feasibility of intelligent service robots that can interact with the world around them. Therefore, additional modules had to

be combined, such as: sensor calibration (RGB-D cameras, laser scanners) and robot modelling (ROS and *MoveIt!*), motion planning (autonomous navigation, arm trajectory planning and manipulation), sensory data acquisition, fusion and processing (for localization and perception), object grasping and manipulation, episodic memorization (Deep-ART) and reasoning (FF-planner) algorithms. These advanced systems were able to deal with the Task Intelligence problem by learning, reasoning and executing specific behaviours based on the environment. This framework was also extended with SLAM algorithms to create semantic maps of the environment, that is, a robust 3D map where all instances of classes are segmented and can be interacted with individually.

The software integration was accomplished under the ROS framework. A large variety of libraries, packages and custom *nodes* were employed to build such complex systems. The experiments were carried out on both a simulated physical environment (Gazebo, Webots) as well as real-life scenarios (kitchens and offices)

This dissertation presents the research conducted during the three years of the PhD program, both at the Technical University of Denmark (DTU) and the Korea Advanced Institute of Science and Technology (KAIST). The results are disseminated into 6 published conference papers (one of them published as a journal article too) and one additional report.

Resumé (Dansk)

Med de hastige fremskridt inden for biomedicin og computerteknologi vil verdens økonomiske og sociale strukturer støde på store udfordringer i de kommende år: på den ene side vil mange jobs blive automatiseret, og selve arbejdet vil flyttes fra lavindkomst manuelt arbejde til højindkomst tekniske jobs. På den anden side vil befolkningspyramiden blive vendt på hovedet og dermed vil arbejdsstyrken falde i forhold til den samlede befolkning. Med de nye forandringer vil der uden tvivl blive behov for udvikling af robotter, som kan forstå og interagere med deres omgivelser; enten til at udføre repetitive og risikofyldte opgaver eller til at tage sig af husholdningen, de ældre eller børnene. Derfor bliver udviklingen af service, intelligente robotter, mere og mere en nødvendighed frem for luksus. Disse robotter bliver nødt til at kunne bevæge sig, opfatte og interagere med den omkringliggende verden.

Hovedformålet med denne forskning har været at udvikle en generel sansningspipeline for robotplatforme til at genkende, detektere og segmentere objekter i dynamiske og komplekse omgivelser uden a priori viden. Derfor blev synergien mellem Machine Learning (Deep Neural Networks) og 3D segmenteringsmetoder udforsket. Fundamentet, arkitekturen og de matematiske teorier for Deep Learning er blevet vurderet for at finde de mest passende netværksmodeller til at løse objektgenkendelses-, detektions- og segmenteringsopgaver. To state-of-the-art netværksarkitekturer var evalueret, omtrænet og implementeret til adskillige anvendelser: YOLO og Mask-RCNN. De anvendte datasæt var både standard datasæt (COCO, SUNRGB-D, ImageNet) og specialfremstillede datasæt (døre og skuffer, søfartsdata). 2D objektsegmenteringen, udledt fra forward-pass af netværkene, virker som en Region of Interest, der maskerer et udbedret dybdebilled. 3D-punkter blev genereret og placeret inde i det fysiske rum, som objektet udfylder. Støj og baggrundsinformation blev filtreret ved brug af adskillige geometriske segmenteringsprocedurer baseret på objektets labels; disse inkluderer primitive fitting RANSAC, euclidean distance og region growing segmentation. Okklusion og geometriske fejl, når objektets model rekonstrueres, var reduceret med multi-viewpoint perspektiver og punktskysregistrering ved brug af ICP/NICP algoritmer. Dermed var 3D modeller af alle de genkendte klasser udledt samtidigt, i reel tid, med en høj præcision og robusthed. Yderligere information kan udregnes fra de nævnte rekonstruktioner som for eksempel position, orientering, størrelse og grebspunkter. Robottens præ-opførsel blev justeret baseret på det data, som blev udtrukket fra omgivelserne; ellers blev ny opførsel udledt fra tidligere viden.

Sansningspipelinen blev også integreret i et mere komplekst system (fra små mobile til humanoide robotter) for at give et proof-of-concept af, at intelligente service robotter kan interagere med deres omverden. Derfor var det nødvendigt at kombinere yderligere moduler som for eksempel: sensorkalibrering (RGB-D kameraer, laserskannere) og ro-

botmodelleringsværktøjer (ROS og *MoveIt!*), baneplanlægning af bevægelser (autonom navigation, baneplanlægning af robotarm og robotmanipulation), sensordataopsamling, -fusionering og -behandling (for lokalisering og sansning), greb og håndtering af objekter, episodisk hukommelse (Deep-ART) og beslutningsalgoritmer (FF-planner). Disse avancerede systemer kunne løse Task Intelligence problemet ved at lære, ræsonnere sig frem og udføre specifikke opførsler baseret på omgivelserne. Pipelinen var også udvidet med SLAM algoritmer for at skabe et robust 3D-semantisk kort af omgivelserne, hvor alle klasserne er segmenteret og kan interageres med individuelt.

Softwareintegrationen var udført i ROS-frameworket. Mange forskellige biblioteker, pakker og skræddersyede moduler var anvendt til at opbygge sansningspipelinen. Eksperimenterne var udført både i simulerede miljøer (Gazebo, Webots) og i reelle scenarier (køkkener og kontorer).

Denne afhandling præsenterer den forskning, som blev udført under et tre års ph.d.-program både på Danmarks Tekniske Universitet (DTU) og på the Korea Advanced Institute of Science and Technology (KAIST). Resultaterne er formidlet i 6 publicerede konferenceartikler (en artikel er også blevet publiceret som en journalartikel) og en yderligere rapport.

Preface

The project was defined as a collaboration between the Automation and Control Group in the Department of Electrical Engineering of the Technical University of Denmark (DTU) and the Robot Intelligence Technology Lab in the Department of Electrical Engineering of the Korea Advanced Institute of Science and Technology (KAIST). The project was fully funded by DTU.

The project was supervised by:

- Professor Ole Ravn (main supervisor), Department of Electrical Engineering, Automation and Control Group, DTU
- Professor Nils A. Andersen (co-supervisor), Department of Electrical Engineering, Automation and Control Group, DTU
- Professor Jong-Hwan Kim (co-supervisor), Department of Electrical Engineering, Robot Intelligence Technology Lab, KAIST

The thesis comprises a summary of the conducted research throughout the period December 2015 - December 2018, submitted and published as articles to peer reviewed scientific conference proceedings.

Kongens Lyngby, December 2018
Adrian Llopart

Acknowledgements

I would like to thank my main supervisors at DTU, Ole Ravn and Nils Axel Andersen for giving me the opportunity of conducting a PhD research and their support and advice throughout the project period. Likewise, I want to thank Professor Jong-Hwan Kim for the opportunity of conducting research with him and his team at the RIT Lab in KAIST and for granting me access to the *Mybot* humanoid robot. I also want to thank Søren Hansen from DTU for his constant support and help, specially when facing the most difficult technical problems: there was no question he ever left unanswered.

Special thank you to my girlfriend Christina Fast for keeping me alive and kicking during my period in South Korea. Finally, I want to thank my family: Natacha Llopart, Jorge Llopart and Esther Maurin, friends: Mattia Baldini, Antonio Pegalajar, Vignesh Krishnamoorthy and Lev Martinez and DTU colleagues: Dimitrios Papageorgiou, Carlos Corchado, Marie Claire Capolei and Robert Miklos, for their constant support throughout these last 3 years of P.h.D. And, most importantly, for listening to me and supporting me during both pleasant and hard times.

Contents

Abbreviations and nomenclature

This nomenclature covers the thesis dissertation. The nomenclatures of the appended publications may differ.

Abbreviations

3D	3-Dimensional
AI	Artificial Intelligence
BA	Bundle Adjustment
BB	Bounding Box
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRF	Conditional Random Fields
DL	Deep Learning
DNN	Deep Neural Network
FC	Fully Connected
FPFH	Fast Point Feature Histograms
FPS	Frames Per Second
GD	Gradient Descent
GPU	Graphic Processing Unit
HRI	Human-Robot Interaction
ICP	Iterative Closest Point algorithm
IoU	Intersection (area) over Union (area)
mAP	Mean average precision
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
MRF	Markov Random Fields
MSE	Mean Squared Error
NICP	Normal Iterative Closest Point algorithm
NN	Neural Network
ORB	Oriented fast and Rotated Brief
PCL	Point Cloud
RANSAC	Random Sample Consensus
RCNN	Region-based Convolutional Neural Network
ReLu	Rectified Linear Activation Unit

RGB	Red Green and Blue parameters, per pixel, in an image
ROI	Robot-Object Interaction
RoI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Feature
SVD	Singular Value Decomposition
SVM	Support Vector Machine
VO	Visual Odometry

Latin symbols

φ	Activation function $[-]$

Sub- and superscripts

$a_j^l = \varphi\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$	the activation a_j^l of the j^{th} neuron in the l^{th} layer is the result of the *weighted input*(z_j^l) passing through the activation φ. z_j^l is the product of the weights w_{jk}^l and activations a_k^{l-1} added to the biases b_j^l, summed over all k^{th} neurons in the $(l-1)^{th}$ layer.
$I = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$	the *intrinsic* matrix includes the *intrinsec* parameters of focal length (f_x, f_y) and principal points (c_x, c_y).
$E = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{bmatrix}$	the *extrinsic* matrix including rotation R and translation T, such that $E = [RT]$
$_{t,\ t-1,\ t+1}$	a certain time step, one time step earlier and one time step later, respectively

Chapter 1

Introduction

> There has never been a time of greater promise, or greater peril.
>
> Klaus Schwab

1.1 Background

An era of great change is upon us. Whether we realize it yet or not, the world that surrounds us has been spun into a state of constant innovation and development thanks to technological advances occurring almost at a daily pace. It is, indeed, a time of great modernization and creativity where ideas that had been deemed impossible not so long ago, now can resurface and flourish. The future is not so far off anymore.

Known as the 4^{th} Industrial Revolution or the Second Machine Age [1], this constant technological reinvention is characterized by emerging technology breakthroughs in a huge variety of fields: most importantly in robotics, artificial intelligence, quantum computing, autonomous vehicles and The Internet of Things (IoT); but also blockchain technology, nanotechnology, biotechnology, 5G wireless and autonomous manufacturing/3D printing, among others. In other words, it will fuse the physical, digital and biological worlds, and by doing so, it will impact all disciplines, economies and industries.

As stated in the 2018 World Economic Forum, we are on the brink of a highly disruptive phase of technological development, where economic stability, social cohesion and occupations will be challenged. Even though opinions and studies differ, it is clear that a significant amount of jobs will be lost due to future automation. Frey *et al.* [2] study found that around 47% of US employment will be at high risk in the next two decades (Fig. 1.1). However, a 2016 study of 21 Organisation for Economic Co-operation and Development (OECD) countries, found that only 9% of said jobs will become automatable. Thus, it is certain that, independently of the final number, there will be imminent workforce changes and job losses. Despite this, on the positive side, highly qualified jobs will be massively created: dull, repetitive labour will be automated and replaced by more creative roles higher up the value chain (*high-skill/high-pay*).

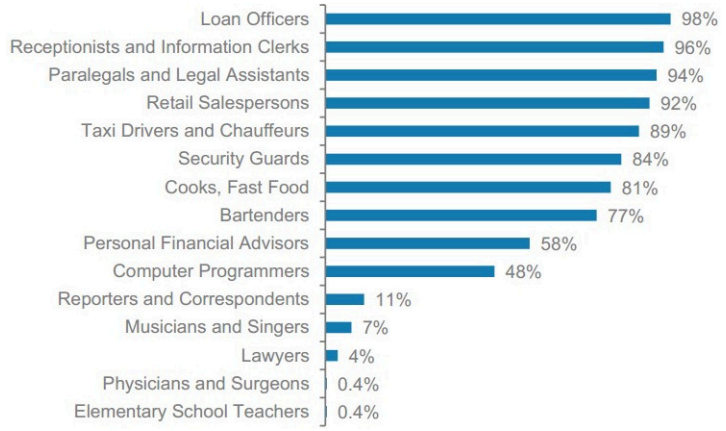


Figure 1.1: Probability of a job being automatable in the next 20 years. Selection of 15 jobs based on the 702 occupations studied by Frey *et al.* (2013) [2]

The potential for raising global income levels and improving the quality of life exists, but humankind will need to be conscientious about the means to accomplish this.

Another way in which the future will be molded is by the usage of Artificial Intelligence (AI) and Machine Learning (ML) techniques, one of the fundamentals in which the 4^{th} Industrial Revolution is based on. Even though most of these methods were developed many years ago, the current upscale of computing power and resources has allowed AI and ML to perform very complex tasks, mainly pattern recognition in application with massive data sets. It is worth noting that AI is the concept of machines being able to carry out tasks in a "smart" fashion, whilst ML is a sub-branch of AI in which machines learn from a given set of data. Some high-profile examples of AI include online targeted advertisement, medical diagnosis, self-driving cars, autonomous navigation (mobile robots or drones), image recognition and segmentation (Deep Learning) and search engines, among others. Only in 2016, 26B$ to 39B$ were invested into AI[1]; where tech giants *Google* and *Baidu* have been estimated to spend around 77% of that amount in R&D. This investment represents a 3X growth with respect to 2013 and is projected to increase at an even higher pace in the following years. The investment in the autonomous driving market alone between 2013-2016 was of more than 80B$[2].

One of the branches of AI that fuses with robotics is called *Cognitive robotics* and it is under heavy development due to the rising needs of the human population for *social* and *service* robotics. The share of the population aged 80 or over is projected to increase from 5.4% in 2016 to 12.7% by 2080, as represented by the population pyramid in Figure 1.2. Not only that, but the number of dependable people (aged below 18 and above 65) will be equal to those in working age. This creates the urge to find means for those dependant people; for that reason, *service and social robots* – robots designed to work alongside humans, are being developed for educational, health care, social care and domestic use. Even though expressing human-like emotions, thoughts or perceiving the world is an extremely complicated domain due to the merging of

[1]source: *Forbes, McKinsey's State Of Machine Learning And AI, 2017*
[2]source: *The Brookings Institution, Gauging investment in self-driving cars, 2017*

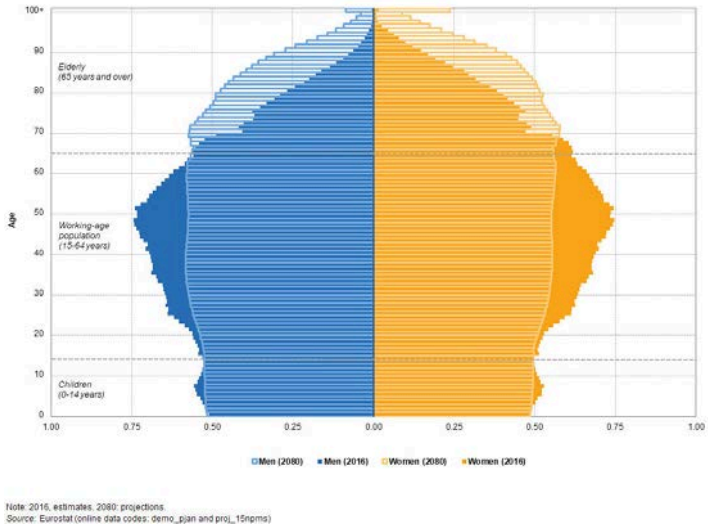


Figure 1.2: Population pyramids of the EU-28 countries, comparison between 2016 and 2080, source: *Eurostat*

diverse fields (technological, social, biological), large sums of money are being invested into these fields. As of the end of 2017, the sales in *service* robots reached the 5.2 B$ mark, with an expected average growth rate of 20 to 25 percent in the period 2018 - 2020[3].

1.2 Motivation, Goals and Scope of the Project

Given the technological advances of the last years, the estimated population aging in the next decades and the new occupational domains, it becomes necessary to create novel methods for automated machines to collaborate with humans and their environment. Therefore, the development of cognitive robotics becomes crucial. The most important challenge in cognitive robotics is the interaction with unknown objects or humans in a complex, dynamic, real world environment, as perfectly explained in Lemaignan *et al.* [3]:

> *Human-Robot Interaction (HRI) is one of the many challenges of Artificial Intelligence (AI). This field lays at the crossroad of several domains of AI and requires to tackle them in a holistic manner: modelling humans and human cognition; acquiring, representing, manipulating in a tractable way abstract knowledge at the human level; reasoning on this knowledge to make decisions; and eventually instantiating those decisions into physical actions both legible to and in coordination with humans. Many AI techniques are invited, from visual processing to symbolic reasoning, from task planning to theory of mind building, from reactive control to action recognition and learning.*

[3] source: *EU-nited, Service Robots - Facts & Figures, 2017*

It becomes evident that to create a machine (robot) that fully understands and handles its surroundings, many engineering fields have to come together and work seamlessly. Precisely, the robot has to be able to, first and foremost, be modelled correctly and have the necessary control mechanisms to make it move efficiently. It then should be able to navigate an unknown environment whilst localizing and memorizing it (SLAM techniques). It must also be capable of semantically segmenting specific objects, locations, humans, etc (depending on the application), extracting important features from each instance and learning them to be able to use them in the future. Finally, the robot has to be *intelligent* enough to reason about all the accumulated knowledge and take the appropriate actions to fulfill a particular task. Thus, building such a complex system becomes such an arduous task that it is necessary to define and shape it into smaller portions.

A clear example, for the reader, of the intended direction of this research project would be a robotic system that is able to correctly navigate a household, absorb all the knowledge that surrounds it and use it to its advantage given a particular task. This could simply be a human asking for a glass of wine; the robot should be able to move to the kitchen area, prepare said glass of wine and take it back to the human.

To this end, the research project, fully funded by the Technical University of Denmark (DTU) and in collaboration with Korea Advanced Institute of Science and Technology (KAIST), aims at developing complex robotic-based systems that fuse current AI trends, traditional control and novel data processing algorithms. The robotic platforms, used as a base to develop new algorithms upon, include small and big mobile robots, robotic arms and a fully fledged humanoid robot. The goals of the project are described as follows:

1. To be able to model and control correctly different types of autonomous systems with a high degree of precision, efficiency and responsiveness (real time).
2. To evaluate state-of-the-art AI and 3D segmentation methodologies. In particular, to implement and enhance pre-designed types of Neural Networks (NN) to suit the needs of specific robotic vision applications.
3. To prove that a seamlessly fusion of 2D and 3D segmentation achieves higher performance than individual or traditional methods.
4. To develop several robotic systems capable of navigating their surroundings, learning and memorizing knowledge over time for future interaction.

These general goals help define the project scope, which includes:

- Mastering of the ROS [4] *Indigo* and *Kinetic* framework, a set of software libraries and tools to help build and enhance robotic applications. Understanding and usage of other important libraries and frameworks like *Moveit!*, *PCL*, *OpenCV*, *Gazebo*, *TensorFlow*, among others, to achieve the set goals.
- Designing, implementing and testing various robotic setups for diverse tasks.
- Robotic modelling and control in the aforementioned platforms. Fusion and processing of the sensory data obtained from the robots. This allows for the general movement of the robot, including navigation, localization and object manipulation.

- Comparison and evaluation between diverse state-of-the-art 3D segmentation methodologies. Idem, with multiple Deep Learning approaches, mainly for object detection and recognition.
- Merging ML and point cloud processing techniques to achieve real-time semantic segmentation, adding knowledge to each segmented instance.
- Robot-Object Interaction with precise object modelling and manipulation.
- Using different memorization mechanisms to allow the robotic systems to obtain and save knowledge from its surroundings for future usage.

1.3 Thesis Outline

The thesis is structured as follows:

Chapter 1 has introduced the necessity of researching about *cognitive robotics* based on the economic and social challenges the future holds and has presented the goals of the project. The literature review and state-of-the-art methods are detailed in Chapter 2, where 3D segmentation techniques, SLAM and Deep Learning methods are discussed. Several Deep Neural Networks models and their applications are considered. Chapter 3 presents a summary of the main contributions of this project, developed as 6 conference papers. Chapter 4 describes in detail the methods used to acquire data and generate 3D point clouds from RGB and depth 2D images. It also deals with the pipeline to segment instances of classes in a cluttered scene, only taking into account the spatial information provided by a sensor (depth image). Thus, algorithms like primitive shape RANSAC, Euclidean Cluster Extraction and Region Growing Segmentation are reviewed. The registration of point clouds is examined using the Iterative Closest Point algorithm (ICP). Finally, this chapter also mentions several SLAM methodologies, some of which have been used throughout the project. Chapter 5 describes the mathematical background of some of the most common DNN architectures used to classify objects in an image. Two major approaches to object recognition, detection and segmentation are outlined: YOLO and Mask-RCNN. The importance of a powerful GPU, a densely annotated dataset and accurate performance metrics for DNN training are also defined in this chapter. The combination of both approaches is dealt with in Chapter 6 which describes the fusion process required to achieve the projects goals. In other words, it showcases a framework where smaller modules add up to a higher level of complexity system: it describes the robotic platforms used, their modelling and control, the libraries employed, the general perception pipeline so that robots understand the dynamic environment around them without any prior knowledge and, finally, the different trajectory warping, object manipulation and data memorization modules necessary to build such complex systems. Finally, the overall conclusions of the project are discussed in Chapter 7, followed by the general direction in which future research could be aimed towards. A list of publications and additional appendices closes this dissertation.

Chapter 2

Literature Review and State of the Art

This chapter presents a concise summary of the most recent and novel research projects, methodologies and approaches relevant to this project. It will be divided into several sections: 3D segmentation (Section 2.1) deals with extracting several features from specific data (images or point clouds) to be able to separate what is relevant from what is not. Deep Learning (Section 2.2), showcases the incredible evolution of Neural Networks (NN) in the last years, and how the development of high performance GPUs has boomed the field of image segmentation and object recognition. Section 2.3 discusses the current approaches for the seamless fusion between ML and 3D segmentation. Finally, Semantic SLAM is examined in Section 2.4. Overall, this chapter should give the reader a basic understanding of the current trends in both domains and help picture how far technology has come; and how much research is still required.

2.1 3D segmentation

One of the main pillars of the project is the segmentation of objects from the environment to allow a robotic system to interact with it. This is achieved using the generated point cloud (PCL) of the environment to segment specific objects from it; in other words, 3D (spatial) segmentation. There are two main approaches that deal with this: **(i)** recognition-based and **(ii)** heuristics-based.

2.1.1 Known objects: recognition-based

Recognition-based methods employ the detection of certain features and their comparison with those of a pre-existing database of known object 3D models. Generating these model databases automatically is described in the literature thoroughly. The ways to obtain a full 3D representation of the model (using point clouds) are divided into 3 groups: **(a.)** rotating a depth-camera around and object [5] [6], **(b.)** mounting the object on a turntable an rotating it in front of a stable camera [7] [8] [9], and **(c.)** having both non-static object and camera [10] (**P2**. [11]).

The extracted features must be reliable, repetitive and highly descriptive to achieve the highest accuracy in the matching process. Some examples of this approach are those presented by [12] [13] [14] [15] [16] [17] [18]. The main problem with this approach is the necessity of a pre-made database to match the features onto. Moreover, detecting and recognizing objects outside the database is impossible, thus limiting the possible applications of such system.

2.1.2 Unknown objects: heuristics-based

The heuristics based approaches try to break down the scene into different Regions of Interest (RoI) and then create higher level hypothesis for every object. A way of achieving this is proposed by fitting primitive shapes (planes, cylinders or boxes) into the segmented point clouds [19] [20] [21] (**P1**. [22]). Some of them also included other methodologies like region growing techniques. Generally speaking, these methods are useful to segment objects in non-cluttered scenes where the object is on top of a table (horizontal plane) or in front of a wall, for instance. However, they perform poorly when trying to differentiate many objects of varying shape and size that are close together.

Another way to segment unknown objects is by applying a graph based approach taking into account both visual and depth features [23] [24]. These often result in 2D Bounding Boxes (BB), a 2D rectangle representing a RoI around the segmented object, which have two main issues: firstly, the relatively low precision causes the generation of poor 3D representations of the objects which, in turn, affect negatively the derivation of grasping poses (based on shape and location of the object) for manipulation. Secondly, it only works when objects are not too close to each other, or else, the BBs would overlap generating very poor results.

The literature proves that segmenting stacked objects is an arduous task in robotic and computer vision. Some authors have dealt with it in diverse ways: the usage of a Support Vector Machine (SVM) to predict surface patches, but failed to join surfaces from the same object [25]; the segmentation of objects based on closed contours and color and 3D information [26], which failed in cases where the objects were small or far away; applying CRFs or MRFs as an energy minimization problem that perform pixel-wise labeling, based on texture cues (normals), in the 2D RGB domain and are then projected to 3D, but produce many classification errors due to occlusions [27] [28]; Asif *et al.* [29] employed a fully unsupervised approach to segment novel objects using a two step method: (i) partitioning the scene into RoI that define the boundaries of all the objects using a novel similarity metric and objective function, (ii) applying a grouping algorithm based on 3D and color cues to generate correct object hypotheses.

2.2 Deep Learning

An additional approach to segmenting objects in cluttered scenes is doing so in the RGB (coloured image) domain. Given a 2D image of a scene, diverse procedures are applied to semantically segment object instances , which are then casted into 3D. By applying this from multiple views, a full, consistent and faithful 3D representation (model) of all objects, is to be obtained.

2.2.1 Early History

The conception, usage and modeling of Neural Networks (NN) is not a novel concept. Back in the 1940's, Alan Turing [30] proposed several criteria to asses if a machine could be determined to be intelligent. However, the first real precursor to neural networks came almost 10 years later with the conceptualization of *perceptons* [31] as an algorithm for supervised learning of binary classifiers. After what is commonly known as the AI winter (all research in NN was halted), a novel idea started taking shape: NN could be constructed from several hidden layers to learn nonlinear functions and could be trained by *backpropagation* [32]. That is, using the loss function of the network to back-propagate the errors and update all the parameters in the model. This then lead to the development of the first CNNs to recognized handwritten digits [33]. Yann LeCun *et al.* continued improving this concept and ended up developing the first ever CNN, *LeNet-5* [34]. This network assumed features were distributed across the entire image and that by applying convolutions, the network could extract them using very few parameters. This contrasted with preceding approaches that employed individual pixels as inputs which lost feature correlations throughout the image. It must be noted that GPUs were nonexistent at that time and that CPUs were terribly slow, that is why using less parameters influenced inversely proportionally the overall performance. Additionally, *LeNet-5* introduced the idea of 3 layer sequences: convolution, pooling and non-linearity (*tanh* or *sigmoid* functions), subsampling using spatial average of maps and using a multi-layer neural network (MLP) as a final classifier.

In 2006, Hinton *et al.* [35] proposed adding and training more and more layers to the NN in a *greedy* fashion. This allowed to model and train NN that were deeper than ever before. For this reason, Neural Networks (NN) were rebranded as Deep Learning (DL). However, this was still not enough since the processing power at that time was limited and so the obtained results were still sub-optimal. This changed in 2012 - Krizhevsky *et al.* [36] introduced the possibility of running larger NN models on GPUs (AlexNet) and, by doing so, halved the state-of-the-art error results in the ImageNet Large Scale Visual Recognition Challenge(ILSVRC), a database for image recognition by Stanford [37]. Other methods, still widely used today, were also introduced, like the *Dropout* and *Rectified Linear Activation Unit (ReLu)* layers. It became evident from then on that DL would overtake classic computer vision and would become irreplaceable for many years, since it had become possible to train very deep neural networks (DNN) instead of just shallower ones. Additionally, AlexNet was commonly recognized to be the reason for current DL's hype and mainstream usage. Less commonly known is the Cireşan *et al.* [38] paper, which also proposed, two years earlier, using GPUs to run NNs with forward and backwards propagation passes for up to 9 layers.

2.2.2 DNN architectures for Object recognition and detection

After *AlexNet's* breakthrough and success, the deep learning research skyrocketed fast until today's state of the art. *Overfeat* [39] proposed, for the first time ever, that a DNN should learn Bounding Boxes, in addition to classes. This allowed for networks to not only recognize an object in an image, but also detect its position.

Opposed to *LeNet-5*'s concept of using large convolutions to capture similar features across the entire image or *AlexNet's* 9x9/11x11 filters, VGG networks [40] reduced drastically the size until 3x3 convolutional layers. This is because it was proven that

when structured in sequence, these smaller filters obtained the same results as larger 5x5/7x7 receptive fields. Moreover, you can stack these 3x3 filters in parallel (VGG reaches up to 512 3x3 parallel filters) to learn even more complex features and their combinations. However, VGG networks had no regularization methods so training them had to be done per layer, which was computationally very expensive. The network achieved a 7.32% Top-5 error in the ILSVRC 2014 challenge.

To solve this issue, Szegedy *et al.* [41] developed *GoogLeNet*, which followed the *Inception* architecture: it used 1x1 convolutional filters before the time-consuming parallel 3x3 layers to reduce the number of features. This is known as a *bottleneck*, and reduces drastically the operational cost of each processed layer. It obtained a 6.67% Top-5 error in the ILSVRC 2014 challenge. *Inception V2* [42] incorporates factorizing the remaining larger convolutions into 3x3 layers. *Inception V3* [43] redefined the previous two architectures by introducing the concept of *Batch-normalization* which calculates the mean and standard deviation of the feature maps in every output layer to normalize the data. This improves the training since the layers will not have to learn offsets anymore. This model reported a 21.2% Top-1 error and 5.6% Top-5 error in the ILSVRC 2012 challenge. Finally, *Inception V4* [44] modified the core *Inception* module by merging it with a *ResNet* module, which over-complicated the original clean network design but obtained a 19.9% Top-1 error and 4.9% Top-5 error.

Single Shot MultiBox Detector (SSD) [45] localized and classified objects in just one forward pass. It followed the architecture from *VGG*, but substituted the fully connected layers with more convolutional layers. The BB regression was based on the *MultiBox* method applied in *Inception* networks, starting from anchors/priors and trying to regress closer to the ground truth. Since many regions were proposed, the worst ones were pruned applying a threshold to the confidence loss and the IoU. SSD-512 (pixel width and height of input images) achieved 76.8% mAP on Pascal VOC [46] challenge, at 22 FPS. SSD-300 obtained 74.3% mAP at 59 FPS.

The *ResNet* [47] architecture appeared at the same time as *Inception V3*. The key concept was simple: the input to a certain layer would come from adding the output of two successive convolutional layer and from bypassing the input to the next layers. With this, and using the *bottleneck* idea, it became possible to train a network with up to 1000 layers. This model achieved a 3.57% Top-5 error in the ILSVRC 15 competition. An improved architecture is presented in *ResNeXt* [48] by adding the idea of *cardinality* which is branching out the original *ResNet* modules by having more dataflow paths, but with a smaller width. Outstandingly, this approach keeps the number of parameters constant.

A network that used the *bottleneck* idea to improve responsiveness keeping performance constant is *SqueezeNet* [49]. The main modules of the net were divided into *squeeze* (*bottleneck*) and *expand* layers, which reduced computation power but maintained the same feature map size. This network model reported the same Top-1 (57.5%) and Top-5 (80.3%) accuracy in the ImageNet challenge as *AlexNet*, but between 50 and 510 times faster, depending on the compression approach and data type of the input image.

Region Based CNN (RCNNs) [50] have also become extremely popular. The original architecture thrived on finding bounding boxes around instances first, and then trying to classify them, as opposed to previous methods. This creates the problem of selecting huge numbers of regions, but is bypassed by using a fixed selective search algorithm to find around 2000 region proposals. These regions are then warped into a square and fed into a CNN to extract features. These are then input into a Support Vector Machine

(SVM) to classify the object within the region and, additionally, predict four values corresponding to an improved RoI (x and y coordinates for the top left edge of the BB, followed by its height and width, pixel-wise). The amount of region proposals and the selective search algorithm made the method not suitable for real-time applications. To solve these issues, *Fast-RCNN* [51] was developed which fed the full image directly into a CNN (like *VGG* or *ResNet*). This was then used to identify the region proposals, still using selective search, which were pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). With this approach, the image recognition process was sped up more than 20 times because there was no need of feeding 2000 region proposals to the CNN each time. *Faster-RCNN* [52] improved the architecture further by eliminating the fixed selective search algorithm and substituting it with a Region Proposal Network. This change reduced the test-time speed 10 times compared to *Fast-RCNN*. The state of the art concerning Region Based CNNs is *Mask-RCNN* [53], it takes the previously defined architectures and adds a branch to *Faster-RCNN* that outputs a binary mask defining if a given pixel corresponds to a certain class or not. Additionally, the *RoIPooling* module is substituted with a *RoIAlign* layer that is able to interpolate pixels reducing the small misalignment's between RoI proposals and actual objects. The result is the input image with a semantic pixel-wise masking on top, with the corresponding class labels and Bounding Boxes, in real-time. This is, to the best of my knowledge, the peak of object detection and recognition at the time of writing this thesis. *SegNet* [54] also tries pixel-wise semantic segmentation through *encoding* (usually *VGG* or *ResNet*) and *decoding* layers (reversed CNN), but is not comparable at all, performance-wise, to *Mask-RCNN*.

Another architecture, that partitions first the image and proposes RoI, is *You Only Look Once (YOLO)* [55]. The difference with the *RCNN* approaches mentioned before is that *YOLO* is a single CNN that predicts bounding boxes and their class probabilities simultaneously. The image is initially partitioned into a grid, where each partition is treated as a RoI. The CNN detects the class probabilities for each RoI, and those that have values higher than a certain threshold are grouped together. This allows for a much faster test-time than many other approaches, but has the problem of detecting objects which are smaller than one grid cell: if two instances of the same object appear in the same cell, *YOLO* will only detect one of them. *YOLO V2* [56] improves the previous network adding batch normalizations, a high-resolution classifier and anchors/priors. Anchors are hand-made BB that are shaped to represent real-life objects. Instead of the CNNs having to predict the BB from scratch, it can now use these pre-made BB and simply find the offsets between the anchors and the BB of an instance in a certain image. Also, the class prediction occurs at a BB level, and not at a cell level as before. All in all, *YOLO V2* runs at around 90 FPS achieving 78.6% mAP on VOC 2007; whereas *YOLO V1* only obtained 63.4%. *YOLO V2* also achieves 72.9% Top-1 accuracy and 91.2% Top-5 accuracy on the *ImageNet* challenge. The architecture for *YOLO V2* is used to detect objects over 9000 classes using hierarchical classification with a 9418 node WordTree - this is known as *YOLO9000*, getting 19.7 mAP overall. Finally, Redmon *et al.* developed *YOLO V3* [57]. If you have not yet read the paper, I suggest you do, it is very well written and highly entertaining. In this architecture, the softmax layers, that keep class probabilities below 1, are removed. Output labels are now non-exclusive, meaning labels such as *child* or *pedestrian* are not mutually exclusive. The cost function calculation is also changed, the feature extractor is improved and, most importantly, the network now derives predictions at 3 different depths. Comparatively, *YOLO V3*

achieves the same results as *SSD* but around 8% mAP lower than *ResNeXt* for the COCO challenge. In spite of this, this network is at least 3 times faster than any other approach.

2.2.3 Deep Learning applied to other fields

Up until now, all the discussion about DL and NNs revolved around the idea of segmenting classes from an RGB image, which is certainly one of the main focuses of the project. However, prior to fully detailing the state of the art for that specific field, it is worth noting the contributions Deep Learning has had in other relevant topics.

Similarly to detecting household objects in images, Litjens *et al.* [58] present a compilation of the state-of-the-art deep learning methods for medical image analysis. It reviews almost 300 contributions; most of them propose modified CNN architectures that detect the existence of tumors, tuberculosis, nodules and other pathologies from limited datasets of images of CT, MRI and X-ray scans from different body parts.

Speech recognition (or speech to text) is a field that has been extensively research by the deep learning community, but did not boom until 2012. Some famous papers using deep learning methods for audio synthesis include [59], [60] and [61]. Evidently, text translation to other languages falls also into this category. Considering the amount of resources invested by huge companies worldwide, it is no wonder the amount of literature that exists about this topic [62], [63], [64], [65] and [66]. Finally, some papers deal with understanding and answering text questions [67] [68] [69].

Image understanding and captioning is an extremely complex domain where DNN not only must be able to detect and recognize object instances, but must also understand the spatial relationships between those classes to be able to summarize what is happening in an image in a few sentences [70], [71], [72], [73] or be able to answer certain questions related to it [74] [75].

Finally, the field which is being transformed the most, but with the least amount of attention due to its secretive methods, is finance. Examples of these DL techniques are difficult to come by, but mortgage risk [76], stock market prediction [77] [78] and trading [79] are good cases of DL tested on finances.

2.3 Deep Learning fused with 3D segmentation

The literature has been progressively redirected towards a synthesis of Deep Learning techniques (usually CNNs) followed by a 3D segmentation of objects. This general approach produces considerably better results than trying to segment object instances directly from a full scene. The reason for this is that the NNs reduce drastically the search space: not only do CNNs generate BB with a higher degree of precision and reliability, but some of them directly generate the full contour of objects, like *Mask-RCNN*[53], in real time, from multiple views, allowing a 3D model of the object to be created simply by casting the 2D segmentation into 3D.

Lai et al. [80] used HOG-based sliding windows detectors to generate BB and class probabilities which were then merged into a voxelized point cloud. Even though the results were very convincing, the usage of a sliding window approach and hand-made features reduced the generalizability of the framework whilst increasing the computational complexity.

An extension of the previous method was proposed by Llopart *et. al* (**P3**. [81])(**P4**. [82]), where, firstly, Convolutional Neural Networks (CNN) are applied to a live video

stream to generate highly precision 2D BB around objects. Secondly, diverse 3D segmentation algorithms are applied in parallel (these incorporate not only fitting primitive shapes, but also other clustering methods like Euclidean Cluster Extraction, Region Growing Segmentation and Color-based Region Growing Segmentation). Finally, the hypothesis are filtered to remove any noise left. This allowed for a fast and precise segmentation of differently shaped objects, in real-time.

Asif *et al.* also merged many of the aforementioned methods to label unknown objects. In [83] an initial 3D segmentation of the scene through the Hierarchical Point cloud Decomposition algorithm (HPD) was carried out to extract high-level object hypothesis. Then the STEM framework extracted CNN features from the objects which, using a hierarchical cascaded forest, were used to compute the objects labels. In [84], a modified R-CNN was used that learnt multiview point features that were robust to spatial variations. Then the class probabilities at a image- and pixel-level were computed. Semantic labelling was finally achieved integrating the probabilities into a CRF-based inference algorithm.

2.4 Semantic SLAM

Obstacle detection and recognition is a very complicated matter by its own. However, the state of the art has pushed the boundaries further by trying (and accomplishing) the semantic segmentation of full, previously unknown, scenes. Thus, the system is able to navigate an environment, localize itself and generate fully labelled 3D semantic maps. To achieve this, NN models for segmentation and SLAM algorithms are fused together: McCormac *et al.* [85] proposed SemanticFusion, which combines a Deconvolutional Semantic Segmentation Network [86], ElasticFusion [87] (for dense scene reconstruction) and a Bayesian update scheme. Similarily, Nakajima *et al.* [88] assigns class probabilities to regions instead of 3D voxels. The same approach can also be taken with multiple sensor fusion for large, outdoor environments [89]. All of these approaches do indeed generate semantic maps, but cannot deal with object-level instances. This means that the system building the maps will not be able to obtain knowledge from the individual objects (size, shape, color, pose, grasps), thus interaction or manipulation will not occur.

Kowaleski *et al.* [90] presented a similar system as the ones aforementioned, based on *Mask-RCNN* [53] and RTAB-Map (Real-Time Appearance-Based Mapping) [91] [92] [93]. However, it is also able to process and obtain information from individual instances of objects from the semantically segmented full scene.

Finally, other approaches directly feed a DNN with the full 3D point cloud of the environment, which outputs object classification, part segmentation and semantic segmentation. This is the case of PointNet [94], which, again, builds semantic maps but does not tackle the problem of individual instances.

Chapter 3

Summary of Main Contributions

The main contributions of the project have been disseminated into 6 conference articles (Publications P1, P3, P2, P4, P5 and P6), all of which have been published at the time of writing this thesis.

P1 A. Llopart, O. Ravn, and N. A. Andersen, "Door and cabinet recognition using convolutional neural nets and real-time method fusion for handle detection and grasping," 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), 2017.

This article describes the procedure for detecting handles using a small mobile robot with no prior knowledge of the environment. It adds an additional layer to the state-of-the-art methods by employing a CNN to robustly, and in real-time, identify doors and cabinets. The YOLO V1 neural network is retrained on a manually created dataset containing almost 400 images of doors and cabinets. Then, a fusion algorithm is applied to the results from a 3D point cloud segmentation and the computer vision image segmentation to obtain an accurate representation of the handles. Important information is then extracted that helps a robotic manipulator turn said handles and open doors.

P2 A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Autonomous 3d model generation of unknown objects for dual-manipulator humanoid robots," Robot Intelligence Technology and Applications 5 Advances in Intelligent Systems and Computing, p. 515–530, 2018.

This article addresses the requirement of certain 3D object segmentation techniques for a prior 3D model of the object to be detected. Using a humanoid robot, or any dual arm setup, it is possible to generate the full mesh of a grasped object. This is achieved through rotating the object and changing it from one hand to another to generate additional viewpoints leading to less occlusions in the final model. This approach merges the two main approaches to object modelling, that is, **(a.)** static objects with movable camera around it and **(b.)** static camera with object being rotated. Thus, both object and camera are being moved meaning a correct and highly precise point cloud registration is needed through the FPFH and NICP algorithms.

P3 A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Generalized framework for the parallel semantic segmentation of multiple objects and posterior manipulation," 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO),2017.

This article presents a novel perceptual pipeline based on fusing machine learning approaches, namely CNN, and 3D segmentation methods, in parallel, in real-time. The network YOLO V2, trained on the 80 different COCO classes, is used to extract RoI of instances of classes in a scene. Up to 7 different point cloud segmentation methods are applied to the 3D cast of the RoI data. All methods and the registration of each objects point clouds are applied in parallel, allowing a correct 3D representation of all recognized object in the environment. Once a specific instance of an object is selected, multiple grasping poses are generated, which are then successfully manipulated by a humanoid robot.

P4 A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Online Semantic Segmentation and Manipulation of Objects in Task Intel-ligence for Service Robots," 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2018.

This article is an extension of the work published in P3. The perceptual framework is merged with several other systems in a humanoid robot, that is, episodic memory, a sequence generator and a trajectory warping module for pre-learnt behaviours. This allows the robot to understand commands and match them with possible knowledge from previous tasks it has already accomplished (or learnt). If the command does not align perfectly with a learnt task or set of actions, the robot is able to infer a new list of actions based on the information in its memory. Then, the perceptual pipeline allows the robot to semantically understand its environment and extract knowledge from it (which is also stored in memory for later use), and the behaviours required to fulfill the task at hand are warped to match the data obtained by the perception framework.

P5 S. Kowalewski, A. Llopart, and J. C. Andersen, "Semantic mapping and object detection for indoor mobile robots," 2018 International Conference on Robotics and Computer Vision (ICRCV), 2018.

This article describes the full solution for the semantic segmentation of an oject in an unknown environment by a mobile robot. A neural network, namely Mask-RCNN, is trained on a modified SUN-RGBD data set to detect up to 8 different classes of common household furniture. The network was enhanced with a Depth input channel when training, but no improvement on performance was found. Due to having pixel-level semantic results for every object instance, the generated point cloud does not require such an intensive noise filtering. A 2D map is generated by casting the 3D point clouds allowing the mobile robot to navigate autonomously (SLAM). Important information (shape, location , orientation) from all objects is then extracted to allow the robot to approach them individually.

P6 M. Blanke, S. Hansen, A. Llopart, J. D. Stets, T. Koester, J. E. Brøsted, N. Nykvist and J. Bang, "Outlook for Navigation - Comparing Human Performance with Robotic Solution," The 1st International Conference on Maritime Autonomous Surface Ship (ICMASS), 2018.

This paper analyzes the possibility of an unattended vessel bridge which, through sensor technology (Radar, AIS, ECDIS and others) and computer algorithms, can obtain the same, and even surpass, human outlook. The assessment is achieved through a wearable eye-tracker on the navigator and a DNN algorithm (Mask-RCNN) retrained on specific maritime classes (sea, land, sky, buoy and ship). The actual comparison is done matching the timestamps at the instant of first observations of a given object for both human and electronic outlook.

Chapter 4

3D imaging and segmentation methods

4.1 Data acquisition

Before defining the methodologies used in this project to process the data, it is important to define how it is obtained and how it is fused with other data streams coming from different sensors. Throughout the implementation of this project, two RGB-D cameras, namely *Kinect V1* and *Asus Xtion*, have been used extensively. These devices obtain RGB and depth images which can then be used to generate 3D representations. Two different sensors are generally included: **(a.)** an RGB color VGA video camera with a pixel resolution of 640x480 and a frame rate of 30 fps, **(b.)** a monochrome CMOS sensor and infrared projector that measure the distance of each point in the environment by calculating the *time of flight* after transmitting invisible near-infrared light and it reflecting on the various surrounding surfaces. The actual depth is calculated by triangulating against a memorized pattern at a known depth.

4.1.1 RGB Image to Depth Image rectification

Before being able to fuse the data from both sensors, the sensors have to be calibrated and rectified, that is, find the transformation matrix between the two sensors reference frames so that there is an exact pixel-to-pixel match between RGB and depth images. Calibration is done using the *Checkerboard* method [95], which finds the focal length, distortion parameters, and lens center of the camera, to approximate the camera as accurately as possible to the *pinhole* camera model (Figure 4.1). This is done for both the RGB and the IR (depth) sensors and results in the discovery of the *intrinsec* parameters, and therefore, the *intrinsec* matrix I. Then, the 6 DoF transformations between RGB and depth cameras reference frames with respect to the world frame are also found (*extrinsic* parameters). All in all, it is now possible to directly map RGB values onto the rectified depth image (Figure 4.2.a and 4.2.b), which will be used to generate coloured 3D point clouds (Figure 4.2.d).

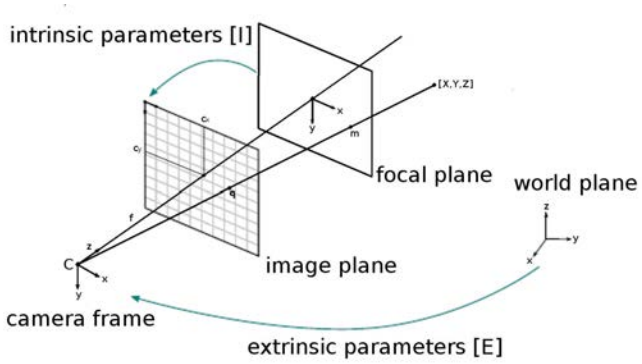


Figure 4.1: Pinhole camera model showing how the *intrinsec* and *extrinsic* parameters are used to change between reference frames.

4.1.2 Casting 2D data to colored 3D point cloud

The conversion from 2D depth data (x, y) to a 3D point cloud (X, Y, Z) is easily achieved once the *intrinsic* parameters, that is, focal lengths (f_x, f_y) and principal points (c_x, c_y), are found:

$$\begin{aligned} X &= (x - c_x) \cdot \text{depth\_image}(x, y)/f_x \\ Y &= (y - c_y) \cdot \text{depth\_image}(x, y)/f_y \\ Z &= \text{depth\_image}(x, y) \end{aligned} \tag{4.1}$$

To generate a coloured point cloud, it is then necessary to find the pixel to pixel correspondence between depth and colored images. Thus, the 3D points $[X, Y, Z]$ are reprojected from the camera frame to 2D pixel coordinates (x', y') using the projection matrix $P = I \cdot E$ (Figure 4.1). This is the multiplication of the *intrinsic* matrix I and the *extrinsic* matrix $E = [RT]$, such that:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = I \cdot [RT] \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = P \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.2}$$

where T and R will depend on the transformation between RGB and Depth cameras. The final 2D pixel coordinates (x', y') are calculated by normalizing u and v:

$$x' = u/w \qquad\qquad y' = v/w \tag{4.3}$$

A new coloured point cloud can be generated based on the new mapping between color and depth images (Figure 4.2.d); however, depth occlusions and colour interpolation (using Nearest Neighbours) result in a coloured point cloud with a certain accuracy loss. The aforementioned procedure is simplified by using the ROS nodelet *depth_image_proc*, which subscribes to the ROS topics containing the *intrinsic* camera information and both the rectified color and depth images. This results in a coloured 3D point cloud ready to be used inside the ROS framework or the PCL library (through *pcl_ros*).

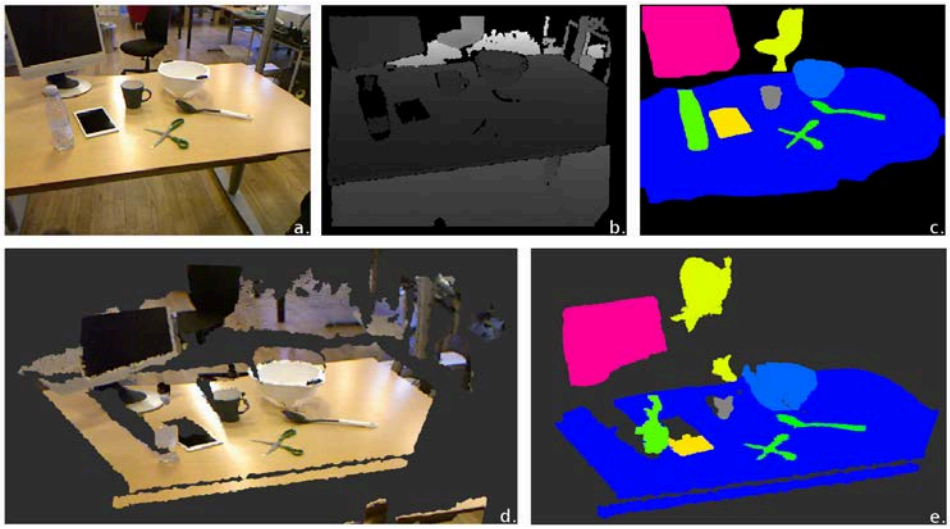


Figure 4.2: Data acquisition and fusion in 2D and 3D. **a.** RGB image, **b.** Rectified depth image, for pixel-to-pixel correspondence with RGB image, **c.** Mask-RCNN results from 2D RGB input image, **d.** 3D point cloud based on the RGB and depth information, **e.** 3D point cloud based on the segmentation procedure using Mask-RCNN.

4.2 Geometrical definitions

3D imaging and segmentation relies heavily on basic Algebra to compute the relationships between different geometric figures in space. The goal is to extract points of interest from a noisy point clouds to be then used for specific purposes. In this section, different methods, used throughout the project, are discussed.

4.2.1 Iterative Closest Point (ICP)

The ICP algorithm was first presented by Besl *et al.* [96]. It is used to find the transformation (rotation and translation, thus 6 DoF) matrix between a *source* and a *target* 2D/3D point cloud, by iteratively minimizing an error metric (usually Mean Squared Error) between matched points from one cloud and the other, to allow the point clouds to be correctly overlapped (Algorithm 1). This approach is also known as point cloud *registration*. This technique is highly widespread among diverse data processing fields such as: shape inspection, motion estimation/tracking, appearance analysis (object recognition), texture mapping/modelling and SLAM algorithms. Convergence, however, will only occur if the starting poses of each point cloud are close enough and the data is not heavily distorted, so the algorithm does not fall into local extremes. **P2** requires the usage of FPFH descriptor matching prior to the NICP algorithm to ensure that both point clouds are already close to each other.

Initially, ICP was developed as a *point to point* registration of a free-form surface. Later, new variants were developed to fit specific needs: Chen *et al.* [97] developed the Normal-ICP (NICP) algorithm that matched *points to planes*, which added surface normal estimation, specially for 2.5D range datasets. Then, Segal *et al.* [98] proposed the

General-ICP to register *plane to plane* datasets, using a probabilistic model to modify the cost function by adding a covariance matrix to each point. Other algorithms were later developed which down-sampled the original point clouds, used K-D data structure trees and better invariant features (SIFT, SURF, ORB) for better point to point correspondence, appropriate error point exclusion methods and improved cost functions.

Algorithm 1 *Iterative Closest Point algorithm*. The pseudo code describes the ICP algorithm used to accurately and reliably register free form surfaces. It finds the rigid transformation H between a target point cloud S and the reference point cloud M that allows for the optimal match [99]. Complexity: $\mathcal{O}(n \log n)$, where n is the number of points in the data set.

Require: A target point cloud S, the coordinates of which fulfill $\{S_i \mid S_i \in \mathbb{R}^3; i = 1, 2, ..., N_S$. A reference point cloud M, the coordinates of which fulfill $\{M_i \mid M_i \in \mathbb{R}^3; i = 1, 2, ..., N_M$.

Output: The rigid transformation H between S and M^k (that is, M after the k^{th} iteration) when the distance between points is lower than a certain threshold τ

for all datapoints i **do**
  for $k \in (0, k_{MAX})$ iterations **do**
    Calculate the corresponding point $M_i^k \in M^k$ in the reference set M so that $\|M_i^k - S_i^k\|$ is minimized.
    Calculate the rotation matrix R^k and the translation vector T^k so that $\sum_{i=1}^{N} \|R^k S_i^k + T^k - M_i^k\|^2$ is minimized.
    Calculate $S^{k+1} = \{S_i^{k+1} \mid S_i^{k+1} = R^k S_i^k + T^k, S_i^k \in S\}$
    Calculate $d^{k+1} = \sum_{i=1}^{N} \|S_i^{k+1} - M_i^k\|^2$
    if $d^{k+1} < \tau \;||\; k > k_{MAX}$ **then**
      return $H = \left(\begin{array}{c|c} R & T^T \\ \hline 0 & 1 \end{array}\right)$
    end if
  end for
end for

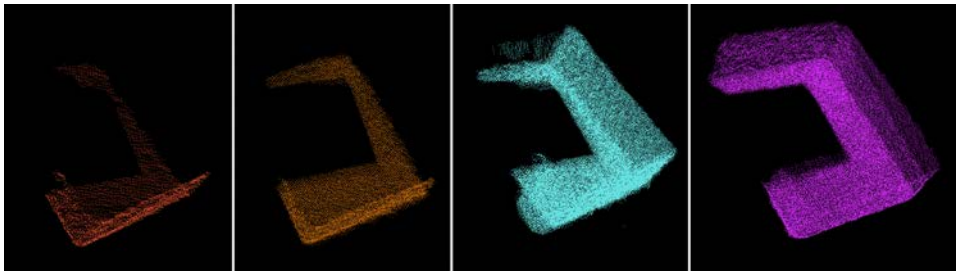


Figure 4.3: Iterative Closest Point algorithm registering different viewpoint point clouds to build 3D models of objects. More details can be found in Publication **P2**.

The original ICP and NICP algorithms are easily accessible in the PointCloud Library (PCL) and have been used considerably in the development of this project due to its good synergy with the ROS framework. These libraries are described in Section 6.3.

4.2.2 Random Sample Consensus (RANSAC)

The Random Sample Consensus algorithm, proposed by Fischler and Bolles in 1981 [100], is used to segment two types of points from a given data set / point cloud (Algorithm 2). The *inliers* will be those points whose distribution fits some set of model parameters, estimated by the algorithm itself. The *outliers* are those points that do not belong to the surface of the model, thus having an error ε larger than a certain threshold τ. Due to processing complexity and real time requirements, the models proposed by RANSAC are usually primitive shapes, such as lines, planes, cylinders and spheres. More complicated models can be estimated but at a computational cost. The functions to be estimated (Θ) and cardinality (parameters to be predicted) of some primitive shapes are as follows:

- Line: $\Theta : b \cdot x + m = y$, cardinality $k = 2$, parameters to be estimated: a, b
- Plane: $\Theta : ax + by + cz + d = 0$, cardinality $k = 3$, parameters to be estimated: a, b, c
- Sphere: $\Theta : (x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$, cardinality $k = 4$, parameters to be estimated: a, b, c, r
- Cylinder: Cylinder model estimation is a bit more complex since a closed form does not exist. Firstly, the plane that fits three sampled points p_1, p_2, p_3 is calculated. Then the circle contained on that plane that goes through all points is found:

$$r = d(p_{i\in(1,2,3)}, c) = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2 + (z_i - c_z)^2} \tag{4.4}$$

  The cylinder model is then estimated with the center c, radius r and normal vector ($\vec{v} = (a, b, c)$) of the plane. The error from any point in the data set will be computed as:

$$\varepsilon = \frac{|\ \vec{v} \times c_p\ |}{|\ \vec{v}\ |} - r^2 \tag{4.5}$$

The same concept of using primitive shapes to build more complex models can be applied to cones and boxes too. It is worth mentioning that the choice of hyper parameters will change drastically the results of this algorithm.

Additionally, when dealing with *table-top* objects - objects found commonly on top of planes (desk, floor), a small algorithm was developed to check whether points were located between the segmented object and the camera, or behind. This was extremely useful in this case since the *inliers* of a plane segmentation are removed and only the *outliers* are saved, but errors might produce points behind the segmented shape which adds considerable noise to the final result. Some results can be seen in Figure 4.4. Once again, these methods are accessible through the PCL Library.

Algorithm 2 *Random Sample Consensus algorithm*. The pseudo code describes the RANSAC algorithm used to iteratively estimate a mathematical model based on a set of observed data to find the best estimated model Θ_{best} that contains as may *inliers* as possible [100].

Require: A reference data set R, the coordinates of which fulfill $\{R_j \mid R_j \in \mathbb{R}^3; j = 1, 2, ..., n_R$. A cardinality k value for the estimated model, i.e the minimum number of points necessary to compute it.

1: **for** $i \in (0, i_{MAX})$ iterations **do**
2: Randomly sample a subset of points from the data R such that $R_s \mid s \in (0, n_{points}) \forall j \rightarrow n_{points}$ is equal to the cardinality k of the model.
3: Compute the model for the selected data Θ.
4: Test all data against the model Θ. *Inlier* points fulfill $\varepsilon_{R_j}^{\Theta} < \tau$.
5: **if** $n_{inliers} > n_{inliers}^{prev}$ **then**
6: Update best model $\Theta_{best} = \Theta$.
7: **end if**
8: **end for**
9: **return** Θ_{best}

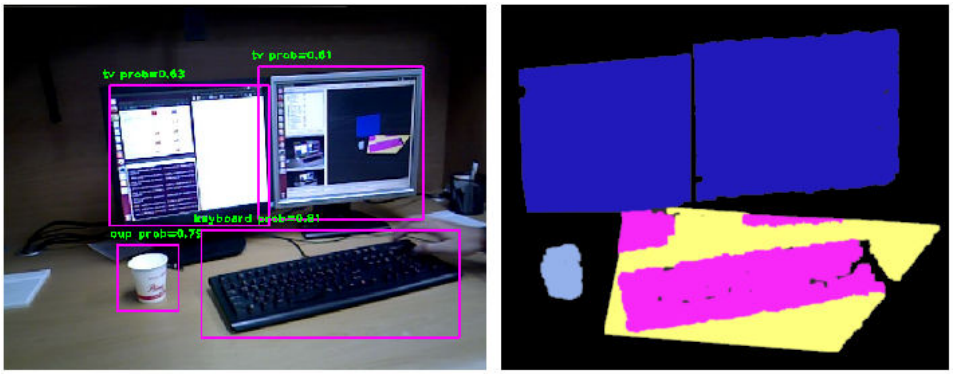


Figure 4.4: **Left:** Input image over-layered with the Bounding Boxes, class labels and probabilities of the YOLO neural network. **Right:** The results are obtained by masking the rectified depth image using the RoIs generated by the network, and then casting the information into a 3D point cloud. If camera calibration is not optimal, the generated point clouds will have a occlusions and background noise that has to be filtered. Thus, the results are then segmented using RANSAC procedures: particularly, the monitors (blue) use a vertical plane segmentation; the keyboard uses a horizontal plane segmentation where the *inliers* correspond to points on the table (yellow) and the *outliers* correspond to points from the keyboard (pink); the glass (grey) uses a cylindrical segmentation. The segmentation errors are solved by multi-view point clouds registration over time using ICP. This procedure is further detailed in Section 6.4.

4.2.3 Euclidean Cluster Extraction

The Euclidean Cluster Extraction (Algorithm 3) segments an input data set into clusters based on the Euclidean Distance, between a point and its Nearest Neighbours, being smaller than a certain threshold τ. It uses an octree data structure to increase performance when finding neighbours (Figure 4.5).

Algorithm 3 *Euclidean Cluster Extraction*. The pseudo code describes the Euclidean Cluster Extraction algorithm used to divide an initial point cloud into smaller clusters based on euclidean distance of nearest neighbours.

Require: An initial data set D, the coordinates of which fulfill $\{D_j \mid D_j \in \mathbb{R}^3; j = 1, 2, ..., n_D$. A maximum distance threshold τ.

1: Create a KD-tree representation for all D_j points.
2: Set up an empty cluster array C and and empty queue Q for all points that have to be checked.
3: **for** all points $D_j \mid j \in (0, n_D)$ **do**
4: Add D_j to current Q.
5: **for** all points $D_j \in Q$ **do**
6: Search all nearest neighbours D_j^{nn} of D_j in a sphere with radius $r < \tau$.
7: **if** D_j^{nn} has not been processed yet **then**
8: Add D_j^{nn} to Q.
9: **end if**
10: **if** all points D_j in Q have been processed **then**
11: Add Q to cluster list C and reset Q to zero.
12: **end if**
13: **end for**
14: **end for**
▷ The algorithm finishes when all points D_j have been processed into a cluster in C
15: **return** set of clusters C

Figure 4.5: Instance segmentation using Mask-RCNN for the class labels and masks and the Euclidean Clustering Algorithm. The segmented classes are table, chair, door and cabinet.

4.2.4 Region Growing Segmentation

Another approach to merging points that share an estimated model is through the Region Growing Segmentation algorithm (Algorithm 4). The result from this algorithm, given an initial point cloud, is a set of clusters where the points of each cluster have a similar smoothness constant (similar angles between their normals).

Algorithm 4 *Region Growing Segmentation*. The pseudo code describes the Region Growing Segmentation algorithm used to segment an initial point cloud into smaller clusters based on a smoothness constraint (angles between point normals). All data points in each clustered region will be part of the same smooth surface.

Require: An initial point cloud P, the corresponding point normals N and curvatures c. A neighbouring finding function $\Omega()$, a curvature threshold c_{th} and angle threshold α_{th}

Initialize the Region list $R \leftarrow \emptyset$.
Initialize the Nearest Neighbours list $B \leftarrow \emptyset$.
Initialize the Available points list $\{A\} \leftarrow \{1, ..., \mid P \mid\}$.
while $\{A\}$ is **not** *empty* **do**
    Current region $\{R_c\} \leftarrow \emptyset$
    Current seeds $\{S_c\} \leftarrow \emptyset$
    Point with minimum curvature in $\{A\} \rightarrow P_{min}$.
    $\{S_c\} \leftarrow \{S_c\} \cup P_{min}$
    $\{R_c\} \leftarrow \{R_c\} \cup P_{min}$
    $\{A\} \leftarrow \{A\} \setminus P_{min}$
    for $i = 0$ to size($\{S_c\}$) **do**
        Find nearest neighbours of current seed point $\{B_c\} \leftarrow \Omega(S_c\{i\})$.
        for $j = 0$ to size($\{B_c\}$) **do**
            Current neighbour point $P_j \leftarrow ()B_c\{j\}$.
            if $\{A\}$ contains P_j **and** $cos^{-1}(\mid (N\{S_c\{i\}\}, N\{S_c\{j\}\} \mid) \leq \alpha_{th}$ **then**
                $\{R_c\} \leftarrow \{R_c\} \cup P_j$
                $\{A\} \leftarrow \{A\} \setminus P_j$
                if $c\{P_j\} \leq c_{th}$ **then**
                $\{S_C\} \leftarrow \{S_c\} \cup P_j$
                end if
            end if
        end for
    end for
    Add current region to global segment list $\{R\} \leftarrow \{R\} \cup \{R_c\}$
end while
return Region list R

The algorithm works as follows: first it sorts all points in data set P based on their curvature c. This is so that the algorithm can start with those points with lowest curvature which belong to flat areas. Then, the first sorted point is selected and added to a set called *seeds*. For every *seed*, the nearest neighbours are found and their smoothness

between normals is evaluated: if its smaller than a certain threshold, the new point is added to the current region. Then, all neighbours are tested for their curvature value; if its less than a certain threshold, the point is added to the *seeds* set. Finally, the current *seed* is removed from the *seed* set. When the *seed* set is empty, that region/cluster is done and the process starts again until all points have been clustered.

An extension of this algorithm is the Color-based Region Growing Segmentation, where the two main differences are: **(a.)** usage of color information instead of normals, **(b.)** usage of a merging algorithm for over- and under- segmentation control, that is, if the number of points in a cluster is smaller than the predefined value, then said cluster is merged with the closest neighbouring cluster.

4.3 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is the methodology used by a robot to move around an unknown environment, localize itself and generate a map (2D/3D), simultaneously. The main issue is the noise in the data extracted from the sensors: friction, control loss or small obstacles can cause bad odometry. Thus, the pose of the robot will always be uncertain, which means the generated map is prone to errors with, at least, that same level of uncertainty. Throughout the last decades, SLAM algorithms have fallen into the following categories: **(a.)** Extended Kalman Filter SLAM (EKF), **(b.)** Sparse Extended Information Filter (SEIF), **(c.)** Extended Information Form (EIF), **(d.)** FastSLAM (v1 and v2), **(e.)** Grid based SLAM and **(f.)** GraphSLAM. The first four fall out of the scope of this project because they are outdated since the new approaches present improved and more accurate results.

4.3.1 Grid based SLAM with Rao-Blackwellized Particle Filters

Grid based SLAM, developed by Doucet *et al.* [101] and Murphy [102], served as the perfect introduction to the development of this thesis since it allowed to add a particle filter approach (artificial intelligence) to solve the pose uncertainty in SLAM, thus being able to generate a 2D ground map only from laser scan data. The RBPF-SLAM algorithm, reimplemented in *MatLab* in Appendix **A1**, generates estimates of the possible robot positions and their potential trajectories (particles) and weights them. The weight indicates how close that particle is to the reality based on scanner data and odometry. Those particles that have lower weights, will die out in the next iteration and, to keep the particle count constant at all times, those particles that have survived will reproduce through resampling stages.

Thus, the goal of RBPF-SLAM is to estimate the joint posterior $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$. In other words, the algorithm should compute the true (or as close as possible) position (x) of the robot and a map (m) for every iteration (on a per particle basis), given solely the laser scan data (z) and the odometry measurements (u). This posterior can be factorized as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{4.6}$$

The first part $p(m|x_{1:t}, z_{1:t})$ is known as the posterior over maps and is computed using the *Occupancy Grid* approach presented by Thrun *et al.* [103], where the generated

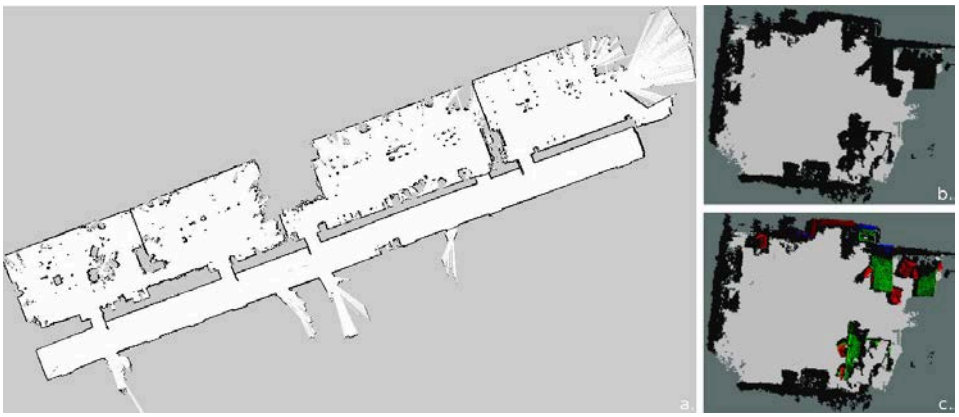


Figure 4.6: **(a.)** 2D occupancy map of the AUT department at DTU, using the ROS *amcl* package based on Grid-SLAM using a Particle Filter, **(b.)** 2D occupancy map of the remaining part of the department, casting the results from P5 onto 2D, **(c.)** the original 3D semantic point cloud viewed from above. A better view is seen in Figure 4.5.

2D map m is partitioned into a grid, where each cell m_i contains a value that refers to the probability of it being occupied $p(m_i) \in (0, 1)$.

The second part, $p(x_{1:t}|z_{1:t}, u_{1:t-1})$, is known as the posterior over potential trajectories and is where the particle filter is applied, where one particle represents a potential trajectory. A particle filter called Sampling Importance Resampling (SIR) is applied, which has 4 main steps: **(a.)** sampling the next generation of particles based on the velocity model probability distribution, **(b.)** importance weighing the particles based on sensory data, **(c.)** resampling new particles and removing old ones and **(d.)** map estimation through the Occupancy Grid algorithm.

An example can be seen in Figure 4.6.a. A more detailed explanation of the Rao-Blackwellized Particle Filter SLAM (RBPF-SLAM) can be found in Appendix **A1**.

4.3.2 GraphSLAM

GraphSLAM [104] improves RBPF-SLAM by being able to retrieve the full path and map, instead of only the current pose and map. This new method requires lower processing capacity whilst having increased accuracy. This is accomplished treating robot poses x_t at a time step t and feature locations $m^{(i)}$ as nodes in a graph. Edges represent measurement constraints bertween two poses or a pose and a feature. Thus GraphSLAM is a global graph optimization problem, which is done with the Maximum Likelihood Estimation (MLE). MLE will estimate the most probable robot poses and features (map) given a set of motion (control command u_t) and sensory measurements (z_t). We then have that the measurement estimate $\bar{z_t}$ and the motion estimate $\bar{x_t}$ are, respectively:

$$\bar{z_t} = x_t + m_t^{(i)} \tag{4.7}$$

$$\bar{x_t} = x_{t-1} + u_t \tag{4.8}$$

Since these estimates have a gaussian probability distribution, then:

$$p_m(z_t) = \frac{1}{\sigma_u\sqrt{2\pi}} e^{-(x_t-\bar{x_t})^2/2\sigma_u^2} \tag{4.9}$$

$$p_u(x_t) = \frac{1}{\sigma_m\sqrt{2\pi}} e^{-(z_t-\bar{z_t})^2/2\sigma_m^2} \tag{4.10}$$

To solve the optimization problem, matrices and covariances must be used. Defining a set of constraints v_t and w_t:

$$v_t = z_t - h(x_t, m_t) \tag{4.11}$$

$$w_t = x_t - g(x_{t-1}, u_t) \tag{4.12}$$

and the formula for the sum of all contraints is:

$$J_{GraphSLAM} = x_0^T \Omega x_0 + \sum_t (w_t^T * R_t^{-1} * w_t + v_t^T * Q_t^{-1} * v_t) \tag{4.13}$$

where $h()$ and $g()$ are the measurement and motion functions and R_t and Q_t are their respective covariances. To create the graph, GraphSLAM is commonly divided in the literature into 2 parts: *front-end* and *back-end*: **(a.)** The *front-end* uses the odometry and sensor measurements to construct the graph by adding nodes and edges iteratively, thus detecting if features from the scene have been previously observed. **(b.)** The *back-end* uses the constructed graph with its constraints and predicts the most probable robot poses and map features locations; thus, solves an optimization problem.

4.3.2.1 RTAB-Map

Real-Time Appearance-Based Mapping (RTAB-Map) is a RGB-D Graph-SLAM approach based on a *loop closure* detector (Figure 4.7). The *loop closure* detector evaluates if the frames obtained with an RGB-D sensor, like a Kinect or an Asus Xtion, represent a new location or somewhere already visited before. As more frames are acquired, more time is required to make the comparison; hence, the complexity is linear.

For this algorithm, the *front end* can use odometry from wheel encoders, IMUs or visual odometry. Visual odometry (Section 4.3.2.3) is done by matching SURF features from two consecutive images. Geometric constraints are obtained with either an RGB-D sensor or from stereo vision. A laser range finder can also be used to increase precision. The relative transform between point clouds for every time step is derived using RANSAC (Sect. 4.2.2) and the corresponding 3D features. Graph management and node creation are also done in this step. For every node, an RGB image, a depth image and and odometry pose is defined; and the edges are the relative transforms. Loop closure is detected using a *Visual Bag of Words* approach by comparing the new image with all previous ones in the graph. The *front end* works at around 20 Hz.

The *back end* deals, as always, with graph optimization and the generation of both 2D (occupancy grid) and 3D (point cloud) maps. Since detection time increases linearly with the size of the map, quantization and graph optimization techniques, like Tree Based Network Optimizer (TORO) or General Graph Optimization (G2O), are applied to ensure that the loop closure process happens in real time. The *back end* works at around 1 Hz.

4.3.2.2 RTAB-Map Modifications

In Kowaleski *et al.* (**P5**. [90]), RTAB-Map was used next to Mask-RCNN to be able to semantically segment an unknown environment in real time. However, to merge these two systems, RTAB-Map had to be slightly modified.

A second graph was created in parallel, where the RGB images were replaced with the semantically segmented outputs from Mask-RCNN. The initial graph had to be kept because RTAB-Map requires the original RGB images to function properly (correct visual odometry). It was also seen that eroding the semantically segmented images with a 7x7 kernel (twice) before adding them to the second graph generated better results. This is because the number of misclassified points being casted into the background was considerably reduced. Having slightly smaller masks was not an issue because they did not modify considerably the overall shape of an object and, most importantly, the small errors in the mapping were resolved when seen and registered from different perspectives.

Additionally, all points that were classified as *background* by the neural net were never introduced in the graph since their representation had no meaning. Finally, all data points further away than a certain threshold (τ_d >2.5 meters) were immediately discarded since the sensors resolution was relatively poor at that distance, allowing for a better fidelity and quality of the generated 3D maps (Figure 4.7).

Even though the proposed methodology obtains positive results as shown in the paper [90], this approach is unable to perform correctly in a dynamic scenario: for example, when objects are moved between instances of a robot mapping a certain

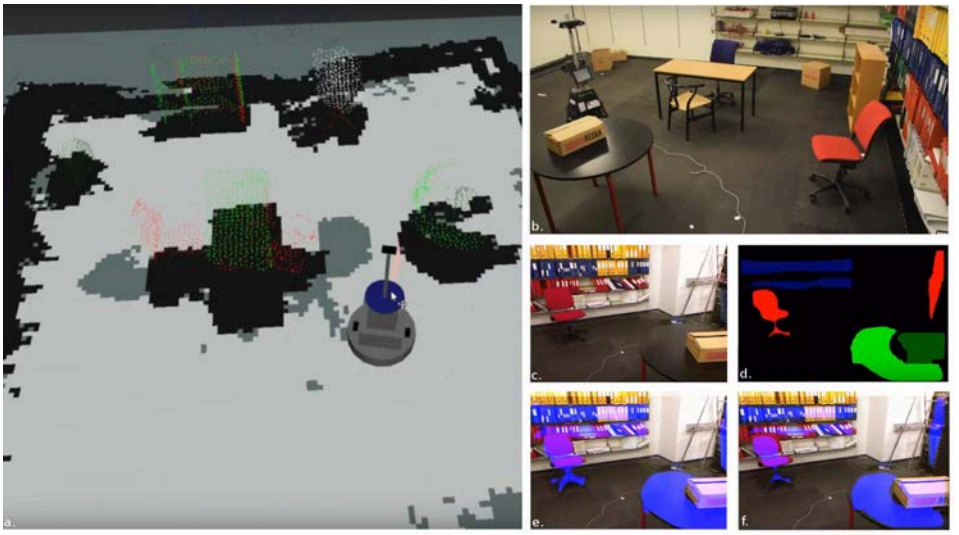


Figure 4.7: Semantic segmentation of an unknown environment integrating RTAB-Map and Mask-RCNN. **(a.)** Generated semantic 3D map, where the instances of every class are highlighted in different colors, **(b.)** Picture of the scene and robot, **(c.)** Live image from the robot, **(d.)** Semantic segmentation using Mask-RCNN, **(e.)** Resulting pixel-wise masks over-layered on top of input image, **(f.)** Erosion of masks for better performance, as demonstrated in Publication P5.

location; or when people walk past the robot as it generates the map. This method is know as *Dynamic-SLAM*: Yang *et al.* [105] use Faster-RCNN (Sect. 5.5.3) to detect potential dynamic objects whilst performing SLAM using the ORB-SLAM2 framework [106] and the TUM RGB-D data set. Scona *et al.* [107] present StaticFusion, a robust and dense RGB-D SLAM method for dynamic environments which detects moving objects whilst simultaneously reconstructing the background structure.

4.3.2.3 Visual Odometry

It has been mentioned previously that Visual Odometry (VO) is an essential component for stereo or monocular SLAM, like RTAB-Map (Section 4.3.2.1), for real-time localization purposes. VO is a technique for estimating camera/robot poses based on images. It can replace other odometries like: wheel odometry (which is less accurate due too wheel slip), GPS, IMU, laser odometry; or it can complement them. To achieve a correct estimation, VO requires a set of input frames from which specific features are detected (commonly SIFT, SURF or ORB). These features are then matched/tracked and the motion of the robot is estimated. Finally there is commonly a local optimization using Bundle Adjustment (BA) to reduce drift.

Particularly for the monocular case, since all robotic platforms developed in this project have employed an RGB-D sensor with just one RGB camera, the set of images at time step ($_t$) is denoted as $I_{0:t} = \{I_0, I_1, ..., I_t\}$. The camera poses at different time instances ($_t$ and $_{t-1}$) are related by the homogeneous transformation H_t:

$$H_t = \begin{bmatrix} R_{t,t-1} & T_{t,t-1} \\ 0 & 1 \end{bmatrix} \tag{4.14}$$

where the set of transformations for all time steps is $H_{0:t} = \{H_1, ..., H_t\}$ and the camera poses are $C_{0:t} = \{C_0, C_1, ..., C_t\}$, such that $C_t = C_{t-1}H_t$.

The main goal of VO is to estimate H_t (Algorithm 5). To do so, it is possible to use appearance-based or feature-based methods. The literature has proven that the latter presents more robust and accurate results. For feature-based methods, the motion can be estimated as a 2D-2D, 3D-3D or 2D-3D match depending on the available data. Since only a monocular camera is employed, all features $f_{0:t} = \{f_0, f_1, ..., f_t\}$ will be 2D. The point coordinates for every matched feature are casted onto a *unit sphere* to attain $p_t = [x_t, y_t, z_t]$ and $p_{t-1} = [x_{t-1}, y_{t-1}, z_{t-1}]$. Exploiting the epipolar geometry [108], we find that:

$$p_2^T E p_1 = p_2^T [t]_x R p_1 = 0 \tag{4.15}$$

where E is the essential matrix, R is the rotation matrix and $[t]_x$ is the skew symmetric matrix, such that:

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{4.16}$$

The essential matrix E is generally computed using a RANSAC based outlier rejection scheme or 5-point algorithm [109], but the the 8-point algorithm [110] can also be used. Then the rotation matrix R_t and translation vector T_t are derived using Singular Value Decomposition (SVD) and the homogeneous transformation matrix H_t between time steps $_t$ and $_{t-1}$ is computed.

The major problem with monocular VO is that absolute scale cannot be computed from two images. For this reason, 3D points are triangulated (intersection of back-projected rays) for each image pair and the relative distances between any combination of 3D points are calculated. The relative scale between image pairs is obtained by computing the mean of the relative distances. Thus, T_t can be rescaled at every time step. The new transformation H_t is multiplied by the last camera pose C_{t-1} to obtain the new camera pose C_t. Finally, windowed Bundle Adjustment is commonly employed to optimize the final camera poses.

Algorithm 5 *Visual Odometry algorithm from 2D to 2D correspondences*. The pseudo code describes the Visual Odometry algorithm used to estimate camera poses based on a sequence of images using 2D feature matching.

Require: A set of images $I_{0:t} = \{I_0, I_1, ..., I_t\}$

1: **for** all images pairs I_t and I_{t-1} **do**
2: Extract and match all features between I_t and I_{t-1}.
3: Compute essential matrix E for all pairs I_t, I_{t-1}.
4: Decompose the essential matrix into R_t and T_t, and form H_t.
5: Compute the relative scale and rescale T_t accordingly.
6: Concatenate the transformations by computing $C_t = C_{t-1}H_t$.
7: Camera pose optimization using BA.
8: **end for**

4.3.3 Additional SLAM methods

Over time, many SLAM methodologies have emerged which try to solve the SLAM problem from different perspectives. Some examples of these approaches are:

PTAM [111] estimates hand-held camera poses in an AR environment. It divides the SLAM problem into two parallel threads (tracking and mapping) to allow state-of-the-art accuracy and robustness at an adequate frame rate. This results in detailed maps consisting of thousands of landmarks.

LSD-SLAM [112] proposes a feature-less monocular SLAM which deals with large-scale maps of the environment in real time. Since it follows the Graph-SLAM logic, it employs a pose-graph of key frames and semi-dense depth maps. Its contributions over its predecessor are a novel tracking method and an accurate probabilistic solution which includes depth noise for tracking.

ORB-SLAM [113] is a feature-based monocular SLAM approach for both small and large indoor environments in real time. The system has loop closure and is robust to motion clutter and re-localization. The camera localization accuracy is much higher than LSD-SLAM and

the overall results obtained are significantly better than other state-of-the-art approaches at the time of its release.

ORB-SLAM2 [106] builds on its predecessor ORB-SLAM by applying Bundle Adjustment (BA) instead of ICP to minimize depth and photometric errors. The usage of close and far stereo-points together with monocular observations allows better results than only monocular. Additionally, this software was released publicly and its API was made very simple to use.

Chapter 5

Deep Learning methods

The beauty of Deep Learning is that it mimics the training and learning process of nature itself. To prove this, the example of a toddler learning how to differentiate farm animals is presented. Initially a toddler does not know what a cow is, and much less the differences between a cow and, say, a pig. For the toddler to learn, an adult must provide an expected output (class label) given a certain input (picture). Thus, the adult will point at a picture of a cow and state that it is, indeed, a cow. This is how the toddler first associates a certain image to a certain label. However, its obvious that no toddler has ever learned farm animals so quickly; its becomes necessary to display many more pictures of cows before the toddler starts extracting features (shape, color, size, etc.) and building relationships between the features and the expected output label. In fact, it is quite possible that the toddler will initially confuse animals (cows and pigs) or forget the names altogether. That is why inputs (many pictures of animals, similar and different) and expected outputs have to be continuous displayed. If the toddler guesses the animals wrong, the adult must tell so, so that the toddler can readjust what he/she understands a cow or a pig looks like.

Likewise, the training process of DNN requires a fully random initialization, feeding it input data (f.ex. images) and waiting for an estimate of the output. Initially the guesses of the DNN will be very incorrect, but by means of the back-propagation algorithm, the difference between the final output (label) and the expected one will go through the network and correct many of the parameters inside it. The problem with this basic approach is that given a labelling error, tuning a specific parameter of the DNN might affect positively future results, but can also mean other parameters are modified, thus reversing the networks performance. That is why it is necessary to find parameter changes that work for all classes simultaneously to achieve optimal results.

Much research has been conducted in this field since it popularized in 2012. The following sections describe the foundations and basic functionalists of simpler neural networks and then more complex architectures, following the structure presented in the book *Neural networks and deep learning* by M.A. Nielsen [114]. Finally, which type of DNN is suited for what purpose in the scope of this project will be discussed.

5.1 Neural Network foundations

5.1.1 Perceptrons

The concept of perceptrons was first developed in 1959 by Rosenblatt *et al.* [31]. This was the initial approach to mathematically modelling the neurons, and their interconnections, that make up human brains. Simply put, a perceptron takes in several binary inputs $(x_1, x_2, ..., x_n)$ and produces a binary output (y). Additionally, the inputs are multiplied by a *weight* factor $(w_1, w_2, ..., w_n)$. All of the inputs are then added together as $\sum_{j=1}^{n} w_j x_j$ and if the value is larger than a certain threshold/bias (θ), then the output (y) would be a binary 1, and vice versa. A perceptron (Fig. 5.1) is thus modelled as follows:

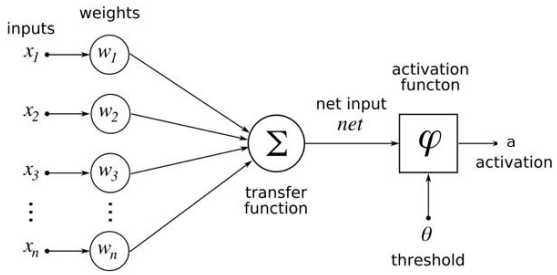


Figure 5.1: Perceptron

$$f(x) = \begin{cases} 0 & \text{if } \sum_{j=1}^{n} w_j x_j \leq \theta \\ 1 & \text{if } \sum_{j=1}^{n} w_j x_j > \theta \end{cases} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{5.1}$$

The second part of Equation 5.1 assumes that $\sum_{j=1}^{n} w_j x_j$ has been simplified to $w \cdot x$ and the threshold θ has been moved to the left side of the equation as a *bias* (b). This second notation is much more common in the literature.

Once a single perceptron has been discussed, it is time to build the neural network. A network of perceptrons (Fig. 5.2) is defined as the interconnection of different perceptrons, where the output of some represent the input to others. How the connections are defined, the number of layers and the types of layer will determine the architecture of the NN. These architectures are known as *Multi-layer perceptrons*(MLP).

The main reason for building a NN is to be able to train it so that it learns specific features from a data set and uses that knowledge to classify it. One would assume that changing specific weight values in certain perceptrons would immediately modify certain aspects of the final outcome. However, the reality is much more complex: changing small parameters to the input of one perceptron will probably make its output change drastically (since it has to be binary). This will propagate through the network, modifying many other values along the way. Thus, the final output of the net will be, generally, something completely different from what was expected and the causes for it will remain untraceable. That is why neural nets are commonly known as *black boxes*, because even though the input values to the NN are known and the produced outcome is visible too, its extremely difficult to trace the reasons for why intermediate layers achieve certain values.

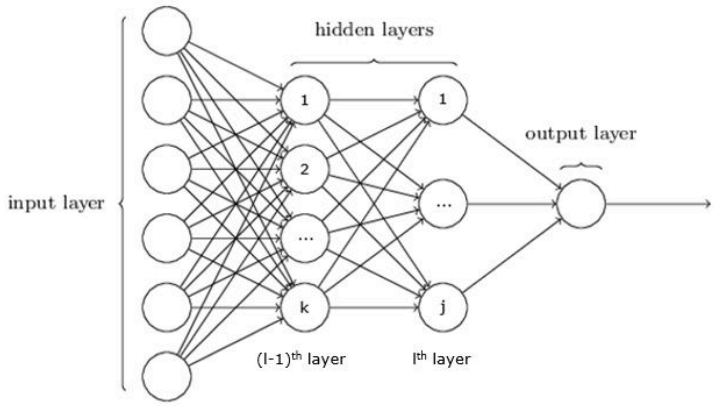


Figure 5.2: Simple neural net formed by layers of interconnected perceptron.

5.1.2 Sigmoid neurons

Thus, to be able to train a NN, its necessary to be able to change some parameters (weights or biases) in certain perceptrons and see the corresponding small change in the final outcome of the NN. If this did indeed occur, it would be possible to fine-tune the NN's parameters based on the difference between the NN's actual and desired outputs. For this reason, *sigmoid neurons* are introduced.

They are very similar to the aforementioned perceptrons, but allow small changes in their weights and biases to have a small and direct change to their own outputs. The only mathematical difference compared to the original perceptrons is the usage of an activation function called *sigmoid function*($\varphi(z)$):

$$\varphi(z) = \frac{1}{1+e^{-z}} \qquad \varphi(x_j, w_j) = \frac{1}{1+e^{-\sum_{j=1}^{n} w_j x_j - b}} \tag{5.2}$$

The smoothness of this activation function (Fig. 5.3) allows small changes in the weights or biases to manifest as small linear changes in the neurons output.

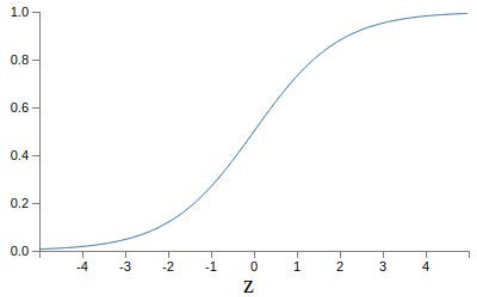


Figure 5.3: Shape of a *sigmoid function*(φ). It is a smoothed out representation of a step function, which is the output of a normal perceptron.

5.2 Defining a Neural Network

As shown in Figure 5.2, a NN has one input layer (for the purposes of this project, it is assumed that the input will always be a grey-scale or RGB image, even though it can be any type of data as described in section 2.2.3), a series of hidden layers (the design and implementation of these will be addressed later) and the output layer (which can include BB values, class label, class probabilities, among others). *Feedforward* networks are neural nets designed so that the outputs of the neurons are used as inputs to the next layers. Another approach are the *Recurrent* networks, which incorporate feedback loops between layers. The following sections describe methods and algorithms that set the foundations on which to build DNN on.

5.2.1 Gradient descent

One of the original approaches to training NNs is using *Gradient Descent* (GD). Firstly, a *cost, loss, error or objective* function is defined which represents the divergence between the actual NN output ($a(x, w, b)$) and the true value ($y(x)$), given x as an input, w as the weights, b as the biases and n as the amount of training inputs. This is also known as the *Quadratic Cost function* or *Mean Squared Error* (MSE), and will be, henceforth, noted as $C(x, w, b)$:

$$C(x, w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a(x, w, b)\|^2 \tag{5.3}$$

If the true output and the produced output are similar, the *cost function* will tend to zero ($C(x, w, b) \approx 0$). The definition of the *cost function* may differ depending on the NN's application. Thus, the training process is simply a minimization of the *cost function*. A way to solve this problem is using *Gradient Descent* (GD). Since the definition of GD for a neural network of hundreds of variables is confusing, for the time being, the GD as a function of just two variables (v_1, v_2) is introduced:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \tag{5.4}$$

if the gradient is defined as:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \tag{5.5}$$

then,

$$\Delta C \approx \nabla C \cdot \Delta v \tag{5.6}$$

For simplicity reasons, $\Delta v = -\eta \nabla C$ is chosen, where η is commonly known as the *learning rate*. Hence, $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$. Since $\|\nabla C\|^2 \geq 0$ and $\Delta C \leq 0$, it is guaranteed that the *cost function* will always decrease. Selecting a correct value of η, also referred to as a *hyper-parameter* in the literature, will modify considerably the NN training's behaviour, from going extremely slow to overshooting completely and "forgetting" the previously acquired knowledge.

The same concepts apply to a multivariable *cost function* like those in NNs. To train a NN and update its variables one must apply the update rule repeatedly:

$$w_j^l \to w_j^{l'} = w_j^l - \eta \frac{\partial C}{\partial w_{jk}^l} \qquad b_j^l \to b_j^{l'} = b_j^l - \eta \frac{\partial C}{\partial b_j^l} \tag{5.7}$$

However, when computing the *cost function*, it is necessary to compute previously all gradients separately ∇C_x for all training inputs and then average them $\nabla C = \frac{1}{n}\sum_x \nabla C_x$, which will require some time. To solve this, *Stochastic Gradient Descent* (SGD) is applied. In this case, the ∇C_x of a random subsample of inputs (*mini-batch*, of size m) is calculated and its average is then considered a good estimate of the real gradient. So the update rule would now incorporate the *min-batches* as:

$$w_j^l \to w_j^{l'} = w_j^l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_{jk}^l} \qquad b_j^l \to b_j^{l'} = b_j^l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_{jk}^l} \tag{5.8}$$

Hence, even though the results from GD and SGD will be slightly different, the important thing is to keep minimizing the *cost function* and not compute the exact value of the gradient.

5.2.2 Backpropagation algorithm

In the previous section, the weight and bias update procedure is described using GD or SGD. However, how to actually compute the gradient of the *cost function* has not been explained yet, that is, $\partial C/\partial w$ and $\partial C/\partial b$. *Backpropagation* is an approach introduced in 1986 by Rumelhart *et al.* [32], which allowed training NN much faster than previous approaches.

To apply *backprop*, two assumptions must be met: **(a.)** The full *cost function*(C) can be written as an average of all *cost functions* for individual training examples (C_x); which is true for equation 5.3. *Backprop* will derive the gradient over each training input, so the total gradient will be found by averaging those values. **(b.)** The cost must be a function of the networks outputs, which is also true.

Backprop will actually calculate first the error δ_j^l, for neuron j^{th} in the l^{th} layer, as the partial derivative of the cost with respect to the *weighted input* (z_j^l):

$$\delta_j^L = \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l}\varphi'(z_j^l). \tag{5.9}$$

Where $\partial a_j^L/\partial z_j^L = \varphi'(z_j^l)$, $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$ and $a_k^{l-1} = \varphi(z_k^{l-1})$. Assuming a cost function C as Eq. 5.3, it is found that $\partial C/\partial a_j^L = (a_j^l - y_j)$. Thus, by rewriting Eq. 5.9 to matrix-form, then:

$$\delta^l = \nabla_a C \odot \varphi'(z^l) = \delta^l = (a^l - y) \odot \varphi'(z^l) \tag{5.10}$$

It is worth noticing that the *elementwise* product of two vectors is denoted as $s \odot t$, such that $(s \odot t)_j = s_j t_j$; this is commonly referred to as the *Hadamard product* or *Schur product*.

If Eq. 5.10 is represented as a function of the next layer ($(l+1)^{th}$), then:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{5.11}$$

By alternating Eq. 5.10 and 5.11 it is possible to calculate the errors for all layers, starting from the end and *backpropagating* through the network.

Two additional equations are now introduced: **(a.)** the rate of change of the cost with respect to any weight in the network (Eq. 5.12) and **(b.)** the rate of change of the cost with respect to any bias in the network (Eq. 5.13).

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{5.12}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{5.13}$$

The mathematical fundamentals of the *backpropagation* algorithm have now been defined. Algorithm 6 describes the step by step procedure to calculate errors, gradients and update the weights and biases from back to front of the DNN.

5.2.3 The Cross Entropy cost function

Learning slowdown is a phenomenon that occurs when training neural networks. It happens when the actual output and the true output are considerably wrong. It has been proven that $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$ are necessary to update the weight and bias values (Eq. 5.7), and both gradients are directly related to the derivative of the weighted inputs to a certain neuron $\varphi'(z^l_j)$ (Eqs. 5.11, 5.12 and 5.13). Thus, the rate at which neurons learn is directly related to $\varphi'(z^l_j)$. Checking Fig. 5.3, it can be seen that in the extreme cases, the *sigmoid function* flattens out, making $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$ smaller and the causing *learning slowdown*.

The *Cross Entropy cost function* for a single neuron is now introduced, to mitigate *learning slowdown*.

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)], \tag{5.14}$$

By substituting $a = \varphi(z)$, applying the chain rule twice and assuming the derivative of the *sigmoid function* is $\varphi'(z) = \varphi(z)(1-\varphi(z))$, the new cost function is simplified to:

$$\frac{\partial C}{\partial w^L_{jk}} = \frac{1}{n}\sum_x a^{L-1}_k (a^L_j - y_j). \tag{5.15}$$

This equation states something obvious, the learning rate is directly proportional to the outputs error, but without any *learning slowdown*, since $\varphi'(z) \equiv a^{L-1'}_k$ has been simplified out of the equation. Similarly:

$$\frac{\partial C}{\partial b^L_j} = \frac{1}{n}\sum_x a^L_j - y_j \tag{5.16}$$

Algorithm 6 *Backpropagation learning algorithm*. The pseudo code describes the algorithm using the Gradient Descent (GD) method per every epoch. When applying Stochastic Gradient Descent (SGD), a mini-batch of m training examples is given, and the *backprop* algorithm is computed for only that mini-batch. Thus, the algorithm will stay the same, but all layer superscripts will not take into consideration the mini-batch: $*_j^l \rightarrow *_j^{x,l}$

Require: NN model with all weights and biases initialized

for x in inputs **do**

FORWARDS PASS

  for j=1,2,...,J nodes in input layer l=1 **do**
    Calculate activations for all nodes: a_j^1
  end for
  for l=2,...,L layers **do**
    for j=1,2,...,J nodes per layer **do**
      Calculate the weighted inputs: $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
      ▷ Index $_k$ for nodes from $(l-1)^{th}$ layer.
      Calculate the activation: $a_j^l = \varphi(z_j^l)$
    end for
  end for

BACKPROPAGATION PASS

  for j=1,2,...,J nodes in the output layer L **do**
    calculate the output error: $\delta_j^L = \nabla_a C \odot \varphi'(z_j^L)$
  end for
  for l=2,...,L-1 hidden layers **do**
    for j=1,2,...,J nodes per layer **do**
      Calculate the node's signal error: $\delta_j^l = ((w_j^{l+1})^T \delta_j^{l+1}) \odot \varphi'(z_j^l)$
      Calculate the the Cost function gradients with respect to weights and biases: $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$
      Update each node's weights and biases: $w_j^l \rightarrow w_j^{l'} = w_j^l - \eta \frac{\partial C}{\partial w_{jk}^l}$ and $b_j^l \rightarrow b_j^{l'} = b_j^l - \eta \frac{\partial C}{\partial b_j^l}$
    end for
  end for

CALCULATE GLOBAL ERROR

  Calculate the Cost Function: $C = \frac{1}{2}\|y - a^L\|^2$

end for

5.2.4 Softmax layer

Another way to alliviate the effects of *learning slowdown* is with the widespread usage of *softmax* layers in the outputs of a DNN. Assuming the *weighted input* for the last layer is defined as $z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$, the *softmax* function is defined as:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}, \tag{5.17}$$

The trick to this formulation is that the sum of all $_j$ neuron activations (a_j^l) in the layer, will equal to 1. Additionally, all activation values will be positive; hence, the *softmax* output can be interpreted as a probability distribution:

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1. \tag{5.18}$$

5.2.5 The Log-likelihood cost function

To prove that the *softmax* layer does, indeed, address the *learning slowdown* problem, a new cost function must be introduced: the *log-likelihood*.

$$C \equiv -\ln a_y^L \tag{5.19}$$

As explained before, it is possible to derive the gradients of the cost function with respect to weights and biases:

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}(a_j^L - y_j) \tag{5.20}$$

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j \tag{5.21}$$

It is worth noticing how Eqs. 5.20 and 5.21 are the same as Eqs. 5.15 and 5.16, respectively, with the only difference of the average over all training instances. Therefore, in general, one would have to choose between the sigmoid output layer and cross-entropy cost function combination, or the softmax output layer and log-likelihood cost function; to ensure that no *learning slowdown* occurs. With the second approach, the output activations can be understood as probabilities. Finally, when applying the *backprop* algorithm using softmax layers and log-likelihood cost, the error formulation has to change to:

$$\delta_j^L = a_j^L - y_j \tag{5.22}$$

5.2.6 Overfitting and regularization

One of the main cruxes of Deep Learning is the trade-off between output accuracy and generalizeability. Given a NN and two sets of images for training and validation, it is possible to identify if the network is *overfitting*, that is, to learn the features of the training set so accurately that the network is not able to generalize anymore. This can be detected when the cost of the training data keeps reducing over time, but the test

data accuracy reaches a "glass ceiling". Another sign is when the cost of the test data, which initially declined as expected, suddenly starts rising again; or that the accuracy of the training set reaches 100%, whilst test accuracy does not. There are several regularization methods to avoid over training, however, all of them cannot be discussed in the scope of this thesis. So, special mentions go to a correct weight initialization, expanding the training data through rotations and distortions and choosing correct hyper-parameters(i.e learning η and regularization λ rates, among others).

5.2.6.1 L2 regularization

One of the most popular regularization methods is *weight decay* or *L2 regularization*, which simply adds a *regularization* term to the end of the unregularized cost function (C_o), where λ is a regularization parameter and n is the size of training data.

$$C = C_o + \frac{\lambda}{2n}\sum_w w^2 \tag{5.23}$$

The factor λ describes if its more important to minimize the unregularized cost function (C_o) or to learn small weights from the regularization term, since small weights means the output of the network will not change too much if certain inputs are changed. And so, a simplified version of the weight gradients with this new cost function is:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}w \tag{5.24}$$

Notice that the regularization term is chosen to not affect the bias because having a large bias doesn't make a neuron sensitive to its inputs. The weight update rule will be the same as GD (and SGD) but with a factor $(1-(\eta\lambda)/n)$:

$$w \to w - \eta\frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n}w = \left(1 - \frac{\eta\lambda}{n}\right)w - \eta\frac{\partial C_0}{\partial w} \tag{5.25}$$

With these regularization changes, it becomes more difficult for the network to learn from input noise and instead learns from patterns scattered across the whole training set, thus generalizing better.

5.2.6.2 L1 regularization

L1 regularization works similarly to *L2 regularization*, where a *regularization term* is added to the cost function. In *L1*, the weights shrink by a constant amount whereas in *L2* they shrink proportionally to w. The rest of the equations are derived as above. *Note:* sgn(w) represents the sign of w.

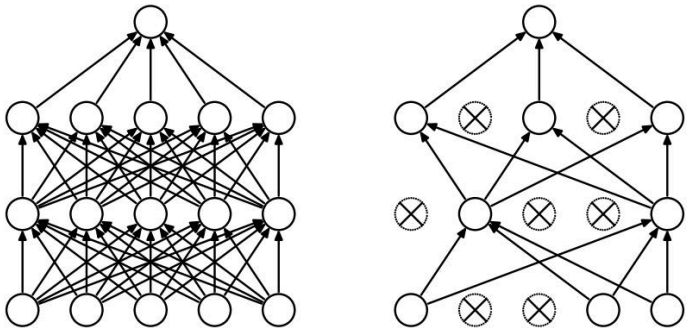
$$C = C_o + \frac{\lambda}{n}\sum_w |w| \tag{5.26}$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}\text{sgn}(w) \tag{5.27}$$

$$w \to w' = w - \frac{\eta\lambda}{n}\text{sgn}(w) - \eta\frac{\partial C_0}{\partial w} \tag{5.28}$$

5.2.6.3 Dropout

This method temporarily disables a certain percentage (usually half) of randomly selected neurons in the network, but leaving the input and output neurons untouched (Fig. 5.4). Then, the input x is forward-propagated through the network and the error is backwards-propagated, as usual. Weights and biases for the modified network are updated and the procedure is restarted with a new input x.



(a) Standard Neural Net (b) After applying dropout.

Figure 5.4: Dropout as proposed by Srivastava *et al.* [115] **(a.)** A standard NN model **(b.)** Modified NN model after applying dropout to roughly half of the neurons.

5.3 Convolutional Neural Networks

Up until now, neural networks have been described in which all k^{th} neurons from the $(l-1)^{th}$ layer were connected to all j^{th} neurons from the l^{th} layer (Fig. 5.2). However, this model structure does not take into account spatial relationships: the interconnection of pixels from opposites sides of an image has the same importance as the relation between neighbouring pixels. This defeats the idea of building a DNN model that uses spatial structure to its advantage. For this reason, *Convolutional Neural Networks* (CNN) are defined and used commonly for image recognition and detection, because they are particularly well-adapted to classifying images based on spatial cues.

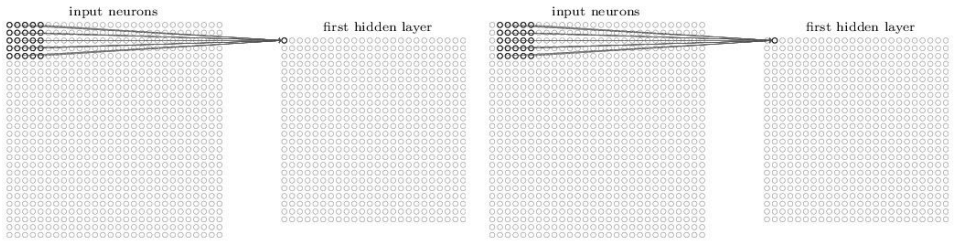


Figure 5.5: Connections between pixels from an image input to the neurons from the first hidden layer. The process will be the same for all other hidden layers of the CNN.

5.3.1 Convolutional layers

The main difference between CNNs and MLPs is that the j^{th} neuron from the l^{th} layer is connected to a whole region (*local receptive field*) of neurons from the previous $(l-1)^{th}$ layer (usually a 3x3, 5x5 or 7x7 group). This region (it might be easier to think of it as a sliding filter) is then shifted throughout the entire layer, generating the inputs for the l^{th} layer (Fig 5.5). The amount of pixels the filter is slid over is called the *stride*.

Another difference is that all local receptive fields will be using the same squared weight matrix (of size *R*x*C*) and bias, hence the term *shared weights* is introduced, which allows to drastically reduce the number of parameters for subsequent layers. This calculation is called *convolution*:

$$a_{j,k} = \varphi\left(b + \sum_{r=0}^{R}\sum_{c=0}^{C} w_{r,c}\, a_{j+r,k+c}\right) \rightarrow \text{simplified: } a^1 = \varphi(b + w * a^0) \tag{5.29}$$

where a^1 and a^0 are the set of output and input activations from a feature map, respectively, and $*$ is the *convolution* operation.

This enables the network to detect the same features over all regions of the input image, thus being able to overcome translational variances. This filter/kernel from layer to layer is commonly referred to as a *feature map* (Fig. 5.6). However, for every layer interconnection, there are typically more than one *feature maps*, and so the output of a convolution is usually a 3-dimensional matrix, where the height and width are dependant on the sizes of the input, the filter and the *stride*, and the depth is the number of feature maps.

5.3.2 Pooling layers

These layers are used intermediately between convolutions to simplify their outputs. It takes all the activations of a previous layer (*feature maps*) and creates more condensed *feature maps* from them. This is accomplished with the commonly used *MaxPooling* layer which takes a region from the input feature map (usually 2x2) and computes its maximum value. This value is then added to the new condensed feature map. At a high level, this layer helps the network focus on the detected features and their interrelation, rather than their specific locations. *L2-Pooling* follows the same approach but uses the square root of the sum of the squares of the region instead of the maximum value.

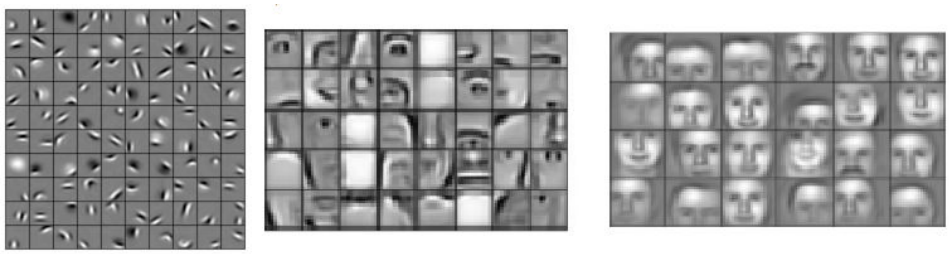


Figure 5.6: Learned *feature maps* at different levels of a CNN trained to detect faces, as presented by Lee *et al.* [116], from simple edge detectors at shallow stages of the network to full face detection at the deepest layers.

5.3.3 Fully Connected layers

The Fully Connected (FC) layers are applied at the end of the CNN architecture and connect every neuron in the last layer to a vector of output neurons representing, usually, the classes detected in the image, much like a MLP. This way, the FC layer is able to learn the non-linearities (combinations) of high-level features coming from the convolutional layers. The *softmax* function can also be applied to covert the output to probabilities.

5.3.4 Rectified Linear activation function

The Rectified Linear Units (ReLU) are commonly used in CNNs instead of *sigmoid* (or *tanh*) activation layers because of proven better performance. The Rectified Linear activation function is very simple:

$$a_j^l = \varphi(z_j^l) = max(0, z_j^l) \tag{5.30}$$

It does indeed outperform *sigmoid* functions because it does not saturate in the limits of z_j^l. There exists very low consensus in the literature as to why this activation function allows for better results; however, after extensive benchmarking, every CNN developer uses them.

5.4 YOLO architecture

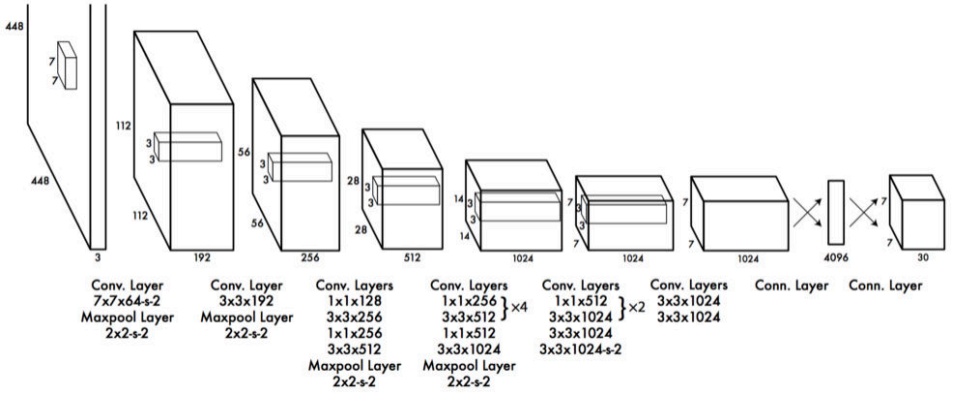


Figure 5.7: Redmons *et al.* YOLO architecture [55].

You Only Look Once (YOLO) is a recognition and detection system based on CNNs. It has 24 hidden layers and 2 FC layers. The convolutional layers use filters of 1x1, 3x3 or 7x7. It differs from other approaches in that it does detection and localization in the same evaluation, thus allowing it to be extremely fast.

The first part of the algorithm is extracting region proposals to be classified. YOLO firstly divides an input image into an **S**x**S** cell-sized grid. Each cell will, hypothetically, encompass only one class, the center of which falls inside that cell. Hence, due to the size of the cells, YOLO is prone to not detecting objects that are too close. Each cell

that includes the center of a class will generate n_{boxes} bounding box predictions and the conditional class probabilities P ($\equiv P_r(class_i) \cdot IoU$) for the whole label set as seen in Fig. 5.8. The BB predictions are x, y, w, h and C, where x and y are the offsets of the cell, the normalized w and h represent the width and height of the BB, respectively; and C ($\equiv P_r(class_i \mid object)$) is the box confidence score on how precise the BB is and how certain it is that it encompasses the class. Thus, the prediction tensor will have a shape of: ($\mathbf{S}$, $\mathbf{S}$, $n_{boxes} \cdot 5 + n_{classes}$), where number *5* corresponds to the vector (x, y, w, h, C). The final class probability is simply the box confidence score times the conditional class probability ($\equiv CxP \equiv P_r(class_i) \cdot IoU$).

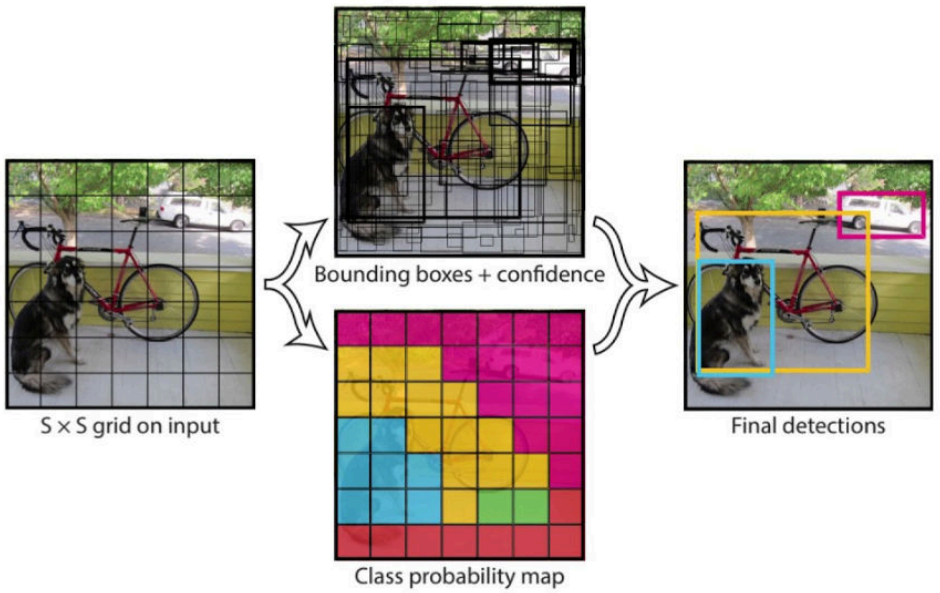


Figure 5.8: Simplified version of YOLO [55] breaking the input image into grids, generating multiple BBs and their box confidence and thresholding the ones with highest class confidence.

The cost function of YOLO is different to what has been explained in this thesis. The final cost function is the sum of a classification cost, a localization cost and a confidence cost. Using the same terminology as in Redmon *et al.* paper [55], the *classification cost* is defined as:

$$C_{class} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (P_i(c) - \hat{P}_i(c))^2 \tag{5.31}$$

where $\mathbb{1}_i^{obj} = 1$ if an object obj appears in a cell i, otherwise $\mathbb{1}_i^{obj} = 0$; and $\hat{P}_i(c)$ is the conditional class probability $\mathbf{P}$ for class c in cell i.

The *localization cost* is a bit more advanced:

$$\begin{aligned} C_{loc} =& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [\ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\ & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [\ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \end{aligned} \tag{5.32}$$

where $\mathbb{1}_{ij}^{obj} = 1$ if the j^{th} BB in cell i is responsible for detecting object obj, else $\mathbb{1}_{ij}^{obj} = 0$; and λ_{coord} is used to increase the weight for the cost in the BB coordinates. Notice that in this equation, YOLO predicts the square root of the width and height to not weight errors of different sized BB equally.

Finally, the *confidence cost* is divided into two: **(a.)** if an object is detected in the box:

$$C_{conf\_detect} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \tag{5.33}$$

and **(b.)** if an object has not been detected in the box:

$$C_{conf\_no\_detect} = \lambda_{no\_obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{no\_obj} (C_i - \hat{C}_i)^2 \tag{5.34}$$

where $\hat{C}_i$ is the box confidence score of the j^{th} BB in cell i, $\mathbb{1}_{ij}^{no\_obj}$ is the complement of $\mathbb{1}_{ij}^{obj}$ and λ_{no_obj} weights down the cost when detecting the background.

Thus, the final cost function C_{YOLO} is the sum of the *classification cost* C_{class} (Eq. 5.31), *localization cost* C_{loc} (Eq. 5.32) and the two *confidence costs* C_{conf_detect} (Eq. 5.33) and $C_{conf\_no\_detect}$ (5.34).

$$C_{YOLO} = C_{class} + C_{loc} + C_{conf\_detect} + C_{conf\_no\_detect} \tag{5.35}$$

Since YOLO can potentially detect twice the same object, a *Non-maximal suppression* algorithm is implemented to remove those lower confidence duplicates and increase the mAP 2-3%. With all of the above, YOLO is indeed a very fast algorithm and much more generizable than other approaches.

As stated in Sect. 2.2.2 YOLOv2 [55] was developed to overcome some of the limitations of the first iteration. Batch normalization was added, which increased the mAP an additional 2%. The training was done in 2 phases, the first one trained for 10 epochs the classifier with high-resolution pictures from *ImageNet*, then, the FC layers were substituted by a convolution layer, and it was retrained end-to-end for object detection. This allowed for a 4% mAP increase. The randomized BB shape proposals were substituted by anchors: BB whose initial prediction represented better the reality. Therefore, instead of predicting BB, the network predicted offset from those hand-made anchors. This reduced the mAP by 0.3% but increased the *recall* by 7%. Finally, the conditional class probability **P** was moved inside the BB prediction tensor (*x, y, w, h,* $P_{\forall classes}$*, C*); which meant that the last layers of the architecture had to be modified accordingly: adding three 3 x 3 convolutional layers with 1024 filters each, followed

by a final 1 x 1 convolutional layer with 125 output channels. Since the FC layers had been removed, the input image was no longer constrained to a certain size anymore. Which, in turn, meant that training images can be of varying sizes, thus acting as data augmentation and making the overall results much more generizable.

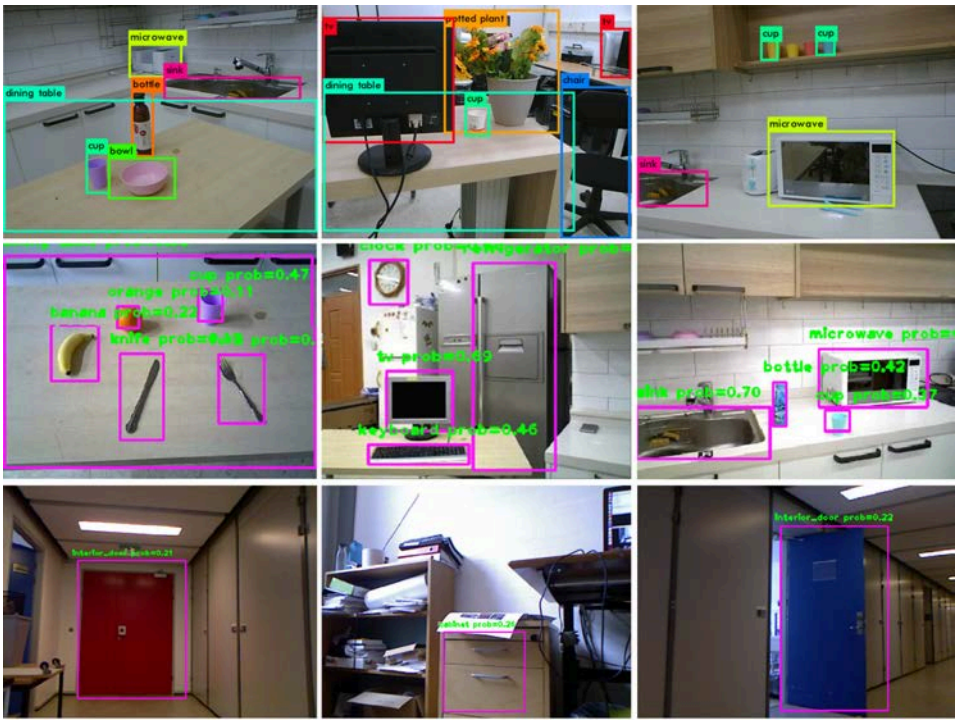


Figure 5.9: Examples of different YOLO results from both off-the-shelf and custom training datasets for diverse classes and scenarios. **Top row:** Images from kitchen and office environments, YOLO trained on COCO dataset; **Middle row:** YOLO integrated with ROS, processing images captured by RGB-D sensor in real time, trained on COCO dataset; **Bottom row:** YOLO integrated with ROS in an office environment trained on a custom doors and cabinets dataset

5.5 Mask-RCNN architecture

Mask-RCNN has gained enormous popularity since it was first published due to its novel idea of segmenting instances of classes in an image and creating binary masks for each one. It is the extension of Fast and Faster-RCNN, thus, to explain Mask-RCNN, its previous architectures must be described. As the name suggests, these algorithms are based on RCNNs, which is an approach (out of several others) to bounding box object detection.

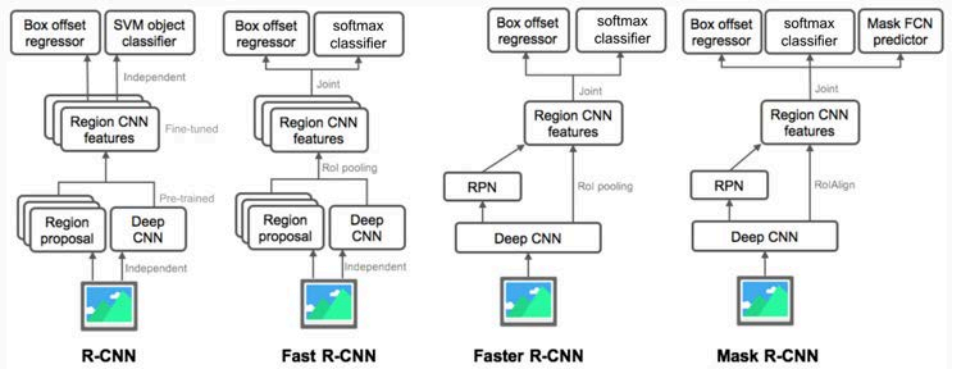


Figure 5.10: Overview of the evolution of the Fast-RCNN, Faster-RCNN and Mask-RCNN network models[1]. Faster-RCNN simply changes the Selective Search algorithm for the Region Proposal Network. Mask-RCNN adds a parallel branch that takes the fixed sized RoI feature maps, processes them through a trained Fully Convolutional layer and outputs semantic instance segmentation; additionally, it substitutes RoI-Pooling with RoI-Align.

5.5.1 Region-Based Convolutional Neural Network (RCNN)

Similarly to YOLO, the RCNN [50] is pretrained as a classifier (VGG or ResNet on Imagenet) first. Instead of breaking the input image down into cells of a grid like YOLO did; in Mask-RCNN, given an input image, RoI are inferred using *selective search* (around 2000 proposals per image), these regions are then warped to fit a fixed input size for the rest of the CNN. The rest of the CNN is fine-tunned with these new classes plus one additional for the "background" (when no class is detected). For this part, a lower learning rate is commonly used. For every RoI, a forward propagation generates a feature vector, which then goes through a binary SVM trained for each class individually. Positive samples are those with an IoU of, at least, 30%. Finally, a regression model is trained to offset the initial RoI for a better BB prediction.

Since there are three independent models (namely, the CNN, the SVM and the regression model), 2000 RoI proposed per every image and a feature vector for every RoI, the process of object detection using RCNN is very slow.

5.5.2 Fast-RCNN

Computation is sped up considerably in Fast-RCNN [51] by merging the three models into one. The CNN feature vector is incorporated to the CNN forward pass for the entire image, so all RoI share this matrix. This is done with a *RoiPooling* layer which converts the RoI features of any size into a small fixed window (the input RoI is divided into a grid and *MaxPooling* is applied to each cell). Finally, the feature vector is then divided into an object classifier (softmax) and a BB regressor. The speed up is considerable, but the real performance bottleneck is the region proposal model (*selective search)* generating a fixed amount of RoIs per input image. This will be changed in the next iteration.

[1] *Source:* https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

5.5.3 Faster-RCNN

In Fast-RCNN [52], the *selective search* is removed and the region proposal algorithm is fused with the CNN architecture as a Region Proposal Network (RPN). The full CNN is pretrained on an image classification task, as done previously. Then the RPN is fine-tuned (where positive samples will be those with IoU $> 70\%$). The RPN works by sliding a window across the input image and predicting BB scales and ratios, thus creating anchors. The rest of the model is trained using the RoI produced by the RPN. The RPN part is then fine-tuned further to be able to share convolutional layers with the rest of the model.

5.5.4 Mask-RCNN

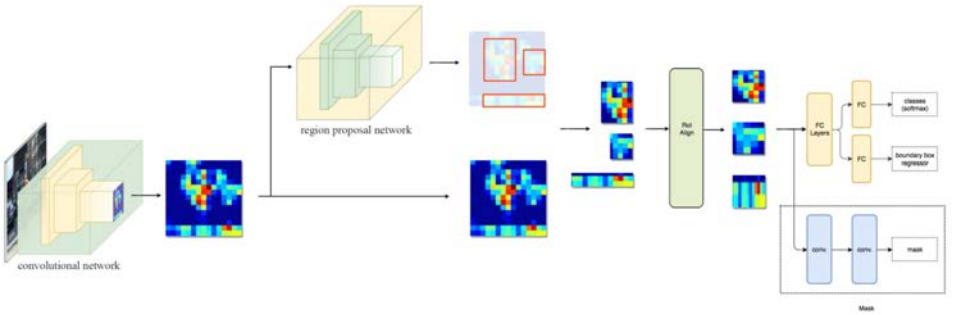


Figure 5.11: Mask-RCNN architecture in detail[2].

The aim of Mask-RCNN [53], implemented using Keras and TensorFlow (a description of these libraries can be found in Sect. 6.3), is to be able to generate binary masks for every class instance in an image. To do so, a third branch is added at the end of the CNN and in parallel with the *softmax* classifier and the BB regressor. This new contribution is a small fully-connected network that predicts mask-segmentation at a pixel-to-pixel level. The issue now is that the network has to be much more precise than before since it now has to work at a pixel level.

The problem with the Region Proposal Network (RPN) generating initially the RoIs is that during *RoIPooling*, *MaxPooling* is applied to downsample the features of a RoI (Sect. 5.3.2), which has the problem of data loss due to stride quantization. Quantaization is the process of rounding up or down the value of the stride for sliding a window given an input feature map and its expected output size (Fig. 5.12). Even though *MaxPooling* introduces invariance to rotations, translations or other distortions, it also generates missalignment and loss of data.

To address this problem, Mask-RCNN introduces the concept of *RoI-Align*, which computes the value of four regularly sampled locations via bilinear interpolation, thus removing quantization at the RoI boundaries (Fig. 5.13). Overall, *RoI-Align* increases the mAP generally more than 5%. Additionally, this method resolves the universal problem of using big strides features.

[2] *Source:* https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272

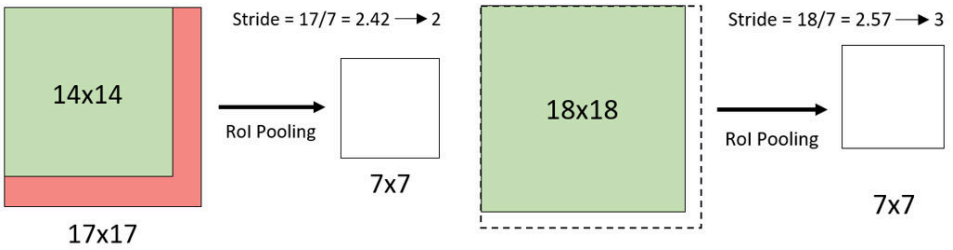


Figure 5.12: When *MaxPooling* a 17x17 input feature map to a condensed feature map of 7x7, its is seen that the stride should be of 2.42, which is then rounded up to 2. This means that the sliding window would only affect a 14x14 sized area and all extra information would be lost. In the inverse case, considering a 18x18 input, the stride would be of 2.57 which would be 3 rounded up, forcing the sliding window to compute non-existent values that would be outside of the size of the input feature map.

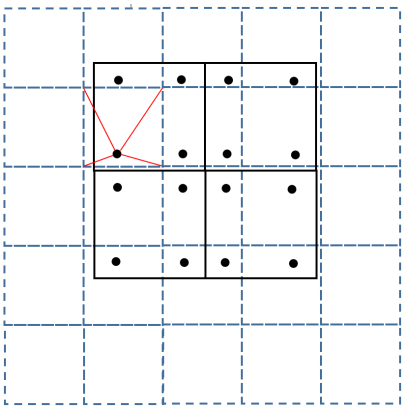


Figure 5.13: The dashed matrix represents the input feature map, and the black matrix, the *MaxPooling* sliding window. The values used in the *MaxPooling* function are derived from bilinear interpolation as shown by the red lines.

The cost function $C_{MaskRCNN} = C_{class} + C_{loc} + C_{mask}$ of Mask-RCNN is a combination of the classification C_{class} (Eq. 5.36), localization C_{loc} (Eq. 5.37) and mask cost C_{mask} (Eq. 5.38) . Thus:

$$C_{class} = \frac{1}{m}\sum_i (-P_i^* \log P_i - (1-P_i^*)\log(1-P_i)) \tag{5.36}$$

where, P_i is the probability of an anchor i being an object, P_i^* is the ground-truth of that probability and m is the mini-batch size.

$$C_{loc} = \frac{\lambda}{n_{\text{anch}}}\sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*), \qquad L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{5.37}$$

where, n_{anch} is the number of anchors, λ helps weight equally classification and localization costs (set to -10 in the paper), t_i is the predicted BB coordinates and t_i^* is the actual ground truth.

$$C_{mask} = -\frac{1}{s_{RoI}^2}\sum_{1\le i,j\le s_{RoI}} \left[y_{ij}\log \hat{y}_{ij}^k + (1-y_{ij})\log(1-\hat{y}_{ij}^k)\right] \tag{5.38}$$

where, s_{RoI} is the size of the generated mask for each RoI, y_{ij} is the label of a cell (i, j) in the ground-truth mask for a specific RoI, $\hat{y}_{ij}^{k}$ is the predicted value for that cell and k is the class number ($k \in 0, K$).

All in all, Mask-RCNN is able to predict pixel-level semantic masks, class label and probabilities for an input RGB image. Some results are shown in Figures 4.2.c, 4.7.d, 5.14 and 5.15.

Figure 5.14: Output predictions for masks, classes and bounding boxes generated by Mask-RCNN on the COCO test set (source: He *et al.* [53]).

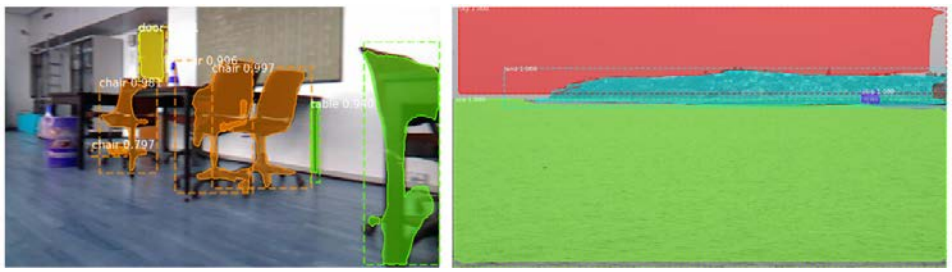


Figure 5.15: Mask-RCNN Predictions for images from: **(a.)** the indoor environment shown in Figure 4.5, **(b.)** a maritime environment, as shown in Publication P6.

5.6 Graphical Processing Unit (GPU)

It has been proved repeatedly in the literature that the computationally intense part of training CNNs happens during the forward and backwards pass, which, essentially is just thousands and thousands of matrix multiplications. The reason for GPUs being so popular after the Krizhevsky *et al.* [36] paper in 2012 was that, even though GPUs were originally designed for better graphical processing (i.e. videogames), it was discovered that they also worked very good on scientific computing. This is because both fields involve applying operations to large matrices.

GPUs have, in comparison to CPUs, hundreds of times much more simpler cores, thousands of concurrent hardware threads, larger memory bandwidth (they are bandwidth optimized instead of latency optimized, as in CPUs), are able to maximize floating-point throughput, and, finally, have a large and fast register and L1 memory. So, all in all, using thread parallelism hides the low latency of GPUs whilst maintaining the large bandwidth memory. Thus, it is proven that GPUs outperform greatly CPUs when training neural nets.

Additionally, the robotic applications developed throughout the thesis required multiple complex systems to work in parallel at a great computational cost. An extreme example of this is the system from Publication P4, where one sole GPU had to sustain: **(a.)** The *Webots* Robotics simulator (robot, environment and physics), **(b.)** A CNN (YOLO) detecting objects in real time, **(c.)** The ROS framework running several programs in parallel, including point cloud segmentation and registration, DNN (Deep-ART) for task selection, full body humanoid control via *RRT**, and, finally, **(d.)** The RVIZ Visualizer, used to display all data exchange between the different programs, the robotic models and all acquired knowledge. For this case, an NVIDIA GeForce GTX 1080 GPU was used. Other projects presented in this thesis have been carried out with the help of lower-end GPUs like the NVIDIA GeForce GTX 1070 or the NVIDIA GeForce GT 730M.

5.7 Datasets and training

Some of the presented projects require the DNN, specifically YOLO and Mask-RCNN, to be retrained to cater to a specific application. When talking about retraining neural networks, it is important to state that generally DNN are not trained from scratch (randomly initialized weights) but, instead, are trained starting from a previous training on another dataset (pre-trained weights). At a high level, all the features the network has learned in its initial layers are surely going to be useful for the initial classification of the new classes, assuming there is some correlation between both datasets. Then, latter layers are fine-tuned to detect the new dataset classes.

In particular, **P1** uses a YOLO architecture (*Darknet* reference model) pre-trained on the COCO dataset [117], and then later fine-tuned for the detection of doors and cabinets. The training set was obtained from *ImageNet*, 510 and 420 images respectively, and was hand labelled: bounding box origin, size and class label. A GeForce GT 730 was employed and achieved 14 FPS for the forward-pass after retraining.

The network Mask RCNN has been also used extensively in this project and has therefore been trained several times. **P5** retrained Mask-RCNN several times with different training sets and hyper parameters. The dataset employed for training was

SUNRGB-D [118] which contained 37 different classes (even though when training this number was reduced to 8 because of overlapping labels) in 10.335 images, all densely annotated: bounding boxes, labels, instance masks. Out of these, 5285 images were selected as training data and 5050 as validation. Finally all masks smaller than a certain threshold were removed. The Mask-RCNN architecture was also modified to include depth data as a fourth input (RGB-D). Due to this, the first convolutional layer of the network had to be resized, thus, the pretrained weights for this layer became irrelevant. This resulted in a network which achieved inferior results than by training only on RGB data. Indeed, the stochastic nature of neural networks and the limited number of trainings is a signal that these results have to be taken with a pinch of salt.

Mask-RCNN was also retrained on a completely novel maritime dataset in **P6**. A set of 200 images were taken using a common DSLR-camera on board a ferry during a span of 3 days, including a foggy day and two sky-clear days.The hand labelling of the images was done using the *LabelMe* software [119]. Masks were generated for every instance, in every image, of the classes *buoy*, *land*, *sea*, *ship*, *sky* (Figure 5.16). 221, 171, 181, 74 and 162 instances of each class, respectively, were found in the dataset. The mean pixel value for the entire dataset had to be calculated to normalize the data inputs to the network. The retraining was done on top of pretrained COCO weights and using an NVIDIA 1080 GTX GPU. The chosen hyper-parameters were: 1000 steps per epoch, 6 classes (the original five plus a background), a learning rate of 0.001 and a learning momentum of 0.9. The *head* of the network (first 3 convolutional layers) was trained first for 10 epochs, then the rest (layer 4 until the end) for another 10 epochs, and finally the whole network for 20 epochs. The final system worked at a frequency of 8 Hz for the forward pass of the network and could semantically segment

Figure 5.16: Hand-made polygon and labels for a custom maritme dataset using the *LabelMe* software

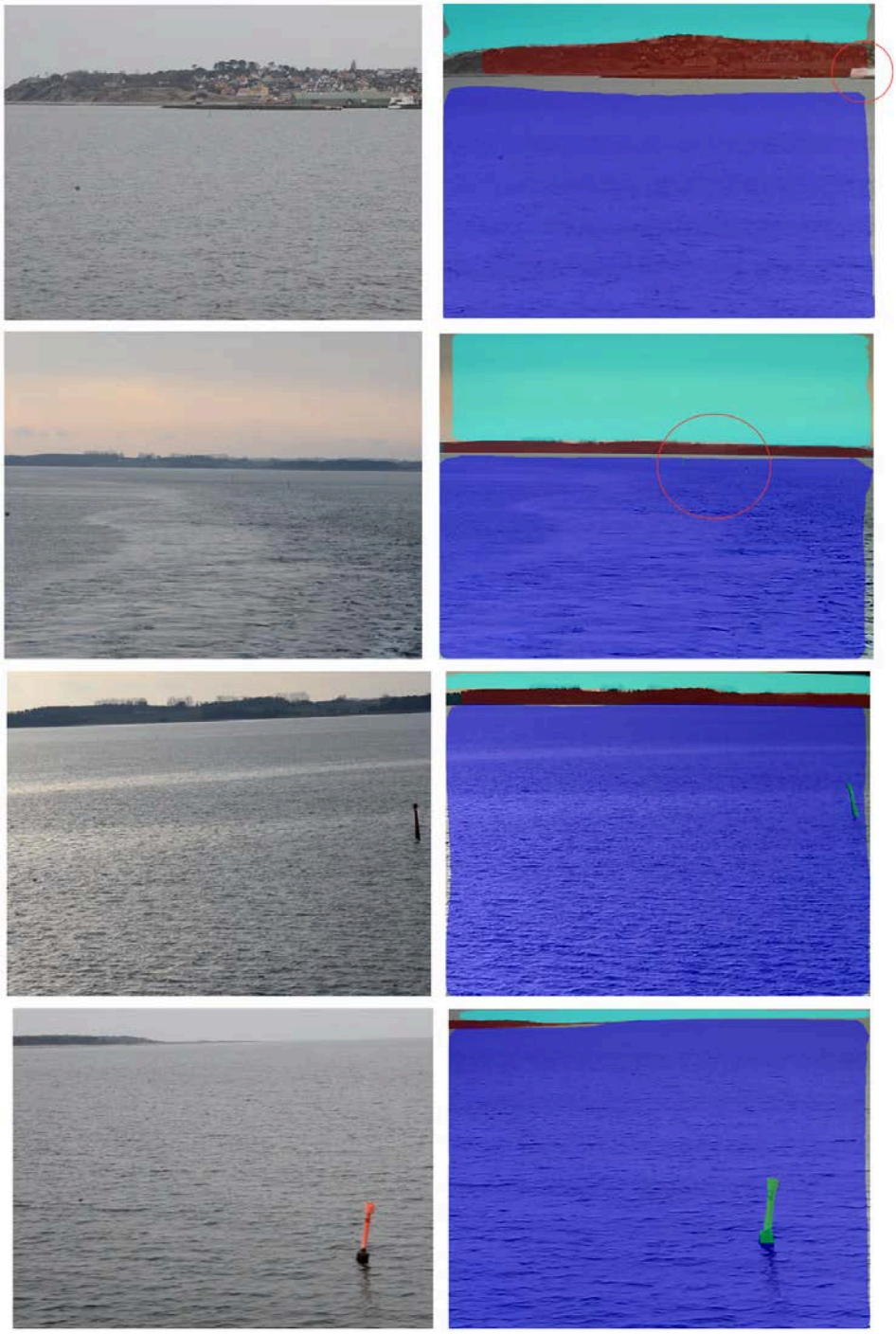


Figure 5.17: Final results of some frames from a video stream, different from the training set

all trained classes even when some of them, specially *buoys* and *ships*, were relatively small (Figure 5.17). On the opposite hand, it was clear that the system overfit due to the limited amount of training data. Additionally, some weather conditions did not appear in the training images, like a cloudy sky, and so the system label those areas incorrectly (Figure 5.18).

Figure 5.18: Results of test images with different weather conditions than the training set. The system fails at recognizing clouds and assumes they are waves in the sea.

5.8 Performance metrics

To evaluate whether a trained network performs correctly or not, a series of performance measures have to be defined. The *mean Average Precision* is the most common metric in the literature for evaluating the results of a classification task; however, a few additional metrics have to be detailed first.

5.8.1 Intersection over Union (IoU)

This metric evaluates how well a prediction matches the ground truth. Simply put, its is the division of the *intersection* and *union* values for a pair of prediction and ground truth. These values are calculated geometrically, depending on if the classification result is a bounding box or a segmentation mask. The *intersection* is the area where they overlap, and the *union* is the addition of both areas.

$$IoU = \frac{intersection}{union} \tag{5.39}$$

5.8.2 Precision

The accuracy of the models predictions is defined by the Precision metric, which is the ratio between the the correct predictions (*tp*) and all predictions. The true positives (*tp*) are those predictions with an IoU > 0.5 for their corresponding ground truths, and the false positives (*fp*) are those with an IoU lower than 0.5.

$$precision = \frac{tp}{tp + fp} \tag{5.40}$$

5.8.3 Recall

A similar metric is the recall, which in turn computes the ratio of correct predictions to all ground truths, evaluating how good the model is at detecting ground truths. The false negatives (*fn*) are defined as ground truth instances that have not been predicted.

$$recall = \frac{tp}{tp + fn} \tag{5.41}$$

5.8.4 Average Precision (AP) and Mean Average Precision (mAP)

The average precision (AP) is calculated by sorting the predictions by confidence level; in some cases, like Mask-RCNN and YOLO, this confidence is already one of the networks outputs. The precision and recall values are then calculated for all predictions and a precision-recall curve is plotted. The curve is interpolated by adjusting the precision value at every recall step (r) to be the *maximum* value for any recall step $r' > r$. The AP is the area below the resulting curve.

This mAP is the mean of the AP for different tests, usually over the whole validation dataset

Chapter 6

Managing the complex systems: merging all modules

The fundamentals of 3D segmentation and Deep Neural Networks have now been established. This chapter now summarizes the elements required to combine DNN and 3D segmentation, detailed in depth in the publications, onto a series of complex robotic systems for real time Robot-Object Interaction. The goals of this project have been established as modelling and controlling (Section 6.2) a series of robotic platforms (Section 6.1) through the ROS framework and additional libraries (Section 6.3). Then, using DNN and point cloud processing (Section 6.4), obtaining data from an unknown environment to be used for object manipulation. This data can also be stored in memory to be retrieved whenever necessary (Section 6.5).

6.1 Robotic platforms used

In the following section, a description of the different robotic frameworks used throughout the project will be presented.

6.1.1 Small Mobile Robot (SMR)

The SMR is a robot developed by the AUT group at DTU Elektro for academic purposes (Figure 6.1.a). It consists of two encoded wheels and several IR sensors, an additional Kinect camera was mounted on it to obtain more sensory data. It runs with a Zotac ION S motherboard and Intel Atom D525 1800 MHz CPU.

All the sensory data and odometry data from the differential drive are obtained by a hardware abstraction layer called the *Robot Hardware Daemon* (RHD). All real-time movement control (and mission control) is done using the SMR-CL language and the Mobile Robot Controller (MRC) in the *MobotWare* framework [120]. *MobotWare* uses RTAI-Linux for real-time performance. It is built on a plug-in based module structure, combined with inter-module communication (TCP/IP sockets). However, due to its limited memory, a modified version of ROS *Indigo* was used when running this robot, which served the same purpose and had the same structure as *MobotWare*, but with many more libraries and functionalites. Thus, specific ROS nodes had to be coded that

would interface with the collected laser scan and odometry data [121]. Additionally all computations had to be outsourced to external computers via wireless or Ethernet because of the robots low processing power.

6.1.2 UR5 robotic arm

Created by the Danish startup company *Universal Robots*, the UR5 (Figure 6.1.b) is a six-axis collaborative lightweight and flexible robotic arm, which weighs only 18.4kg, has a payload of 5kg, reach capabilities of 850mm and a repeatability of 0.1mm, which allows quick precision handling. Most importantly, and this is what sets this robot arm aside from previous models, is that it can work alongside humans because of its automatic brake when colliding with obstacles at rates higher than 200N; thus, there is no need for a safety cage to mark off the its work space (that is the collaborative aspect mentioned before). However, full safety is not guaranteed because the joints have a max speed of 180°/s with the Tool Center point (TCP) being able to move at a maximum of 3m/s. So even though collisions will be detected, the robot may still be harmful. In fact, the most probable damage reason for this arm is self-collision since it has no self-collision prediction.

Singular positions are also a problem since they generate violent movements. Throughout the project, the UR5s firmware has been updated leading to less sudden movements and a more stable configuration in specific singular poses. Additionally, when modelling the robot arm in *MoveIt*, the joint rotation range for some joints was limited to reduce further the self-collision probabilities.

The end effector is interchangeable so that different sensors or grippers are be easily mounted on it. The robot can be manually controlled by a touchscreen but for the purposes of this project, all control was done through the ROS framework and the *MoveIt* libraries, enabling the users to interact with the arm and its extensions simultaneously.

6.1.3 iRobot ATRV-Jr

The *iRobot* company was founded in 1990 to create and develop robotic platforms for space exploration and military defense. Nowadays, the company is mostly famous for the creation of the *Roomba* and other household cleaning devices. Nevertheless, they did produce the autonomous rover *ATRV-Jr* which has been used for navigation in this project (Figure 6.1b).

The robot has a size of 55x77.5 *cm* and weighs 50 *Kg*. It can carry up to 25 *Kg* of payload. It has a bumper sensor to detect frontal collisions and also incorporates a SICK Laser rangefinder, an electronic compass, GPS and a 4-wheel differential drive leading to 3-6 hours of autonomy. Since the laser scanner produced limited results (only 2D), a Kinect camera was added to the robot. Also, to expand the range of applications for this robot, a UR5 robotic arm was mounted on top of it. Additionally, a *Robotiq* 2-Finger Adaptive Robot Gripper and a *Robotiq* Force-torque sensor were mounted on top of the arm too. This created a mobile platform capable of navigating and sensing an unknown environment, and being able to interact and manipulate objects in it. Similarly to the SMRs, ROS *Indigo* was installed to operate the robot instead of *MobotWare*.

Figure 6.1: All robotic platforms used throughout the project: **(a.)** Small Mobile Robot (SMR) created by the AUT group of DTU Elektro, **(b.)** *iRobot* ATRV-Jr mounted with an UR5 robotic arm on top, **(c.)** Guidebot robot and **(d.)** *MyBot* humanoid robot.

6.1.4 GuideBot

The Guidebot (Figure 6.1.c) is a complex AGV (Autonomous Ground Vehicle), developed in 2012 by DTU in collaboration with the *Danish Technological Institute*, capable of traversing intelligently its surroundings, understanding and obtaining knowledge from them and, if necessary, interacting with them. It is a differential drive robot with a collision detection sensor (bumper) around it, a SICK LMS 200-30106 laser rangefinder and two Kinect sensors (facing frontwards and backwards).

This platform was used in the development of the *semantic mapping* task (**P5**), which required a lot of computational power for the computer vision and neural network

execution. For this reason, all data acquired by the robot was sent to an external PC with a GeForce GTX 1080 GPU which had 8GB of the memory and 2560 CUDA cores. The information flow between robot, sensors and external PC is detailed in Figure 6.3.

6.1.5 MyBot

Mybot is a humanoid robot developed at the Robot Intelligence Technology (RIT) Laboratory at KAIST (Figure 6.1.d). Its motherboard is an Odroid XU and even though initially it was set up with Ubuntu 14.04 and ROS *Indigo*, eventually it was upgraded to Ubuntu 16.04 and ROS *Kinetic*. Sensor-wise, it has an Xtion ASUS RGB-D sensor mounted in its head and a laser range finder at the base. It navigates using an omnidirectional three wheel drive, on top of which, a human-like torso, arms and head are mounted. The head includes 2 DoF for panning and tilting, exactly as the torso. Each arm has 7 DoF with 3 finger hands; just 1 DoF to open and close all fingers simultaneously.

The robot has also been fully modelled in *MoveIt* for a full body control and motion planning in the *Webots* simulation environment (Figure 6.2).

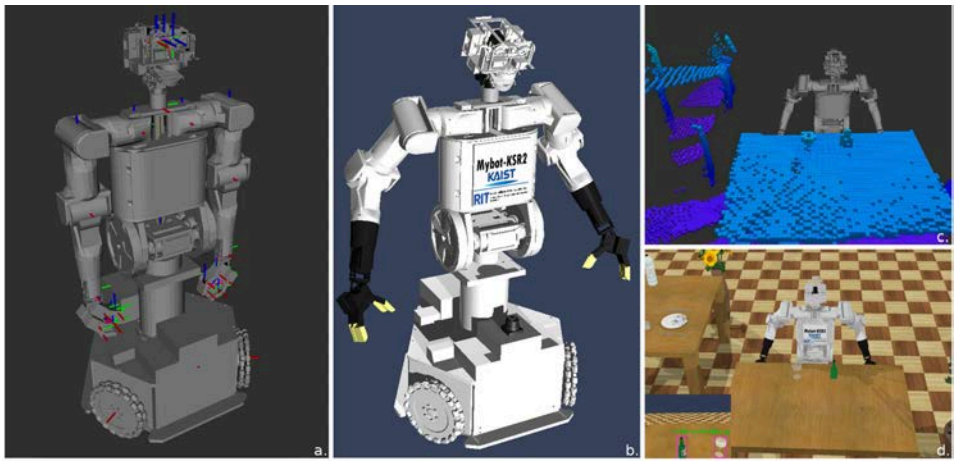


Figure 6.2: *MyBot* humanoid robot modelled in: **(a.)** *RViz*, showing all the joint and sensor reference frames, **(b.)** Meshed model used for simulation, **(c.)** Octomap voxel occupancy grid based on the data from an RGB-D sensor mounted on the head, **(d.)** Webots physical simulator model and environment (used in **P4**).

6.2 Modelling and control of robotic platforms

All robotic platforms were modeled through the *MoveIt!* libraries for a correct pose estimation of all joints and accurate relative transformations between frames. This is done using the graphical user interface of *MoveIt!* where the goal is to generate an Semantic Robot Description Format (SRDF) file. This file includes all the kinematics, inertial, visual and sensing properties of an isolated static robot, which are already defined in the Unified Robot Description Format (URDF), written in an XML format (Code 6.1). An extract of the relative transformations between joint frames for the *MyBot* humanoid robot can be seen in Appendix **A2**.

```
<?xml version="1.0" encoding="utf-8"?>
<robot name="mybot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <link name="base_footprint"/>

  <link name="base_mobile">
    <visual>
      <geometry>
        <mesh filename="package://mybot_description/dae/base_mobile.
            dae"/>
      </geometry>
      <material name="gray">
        <color rgba="0.8 0.8 0.8 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://mybot_description/dae/base_mobile.
            dae"/>
      </geometry>
    </collision>
    <inertial>
      <origin rpy="0 0 0" xyz="0.0 0.0 0.3"/>
      <mass value="30.0"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
          izz="1.0"/>
    </inertial>
  </link>

    <joint name="base_footprint_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_mobile"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </joint>

</robot>
```

Algorithm 6.1: Example of an URDF extract defining the physical model for the robots base and the relative transformation (joint) between two links. The color, mesh/model, mass and inertia are also defined in the URDF.

Hence, the SRDF file is used to view the robot in the visualization tool *Rviz* or allows the robot to sense and physically interact with a simulated environment through Gazebo or Webots or display all sensory data in *Rviz* (Figure 6.2).

Additionally, it also includes information that has a semantic aspect to it: through the *MoveIt! Setup Assistant*, one is able to customize the robots model through: **(a.)** generating the self-collison matrix for all joints. This disables collision checking between those joints that are too far away to every collide, thus reducing drastically the computational time when motion planning. **(b.)** adding virtual joints to attach the robot to the *world*. **(c.)** specifying planning groups (generally as kinematic chains) so that when moving, the robot does not need to plan a full body motion. **(d.)** adding robot default poses. **(e.)** defining end-effectors. **(f.)** adding passive joints.

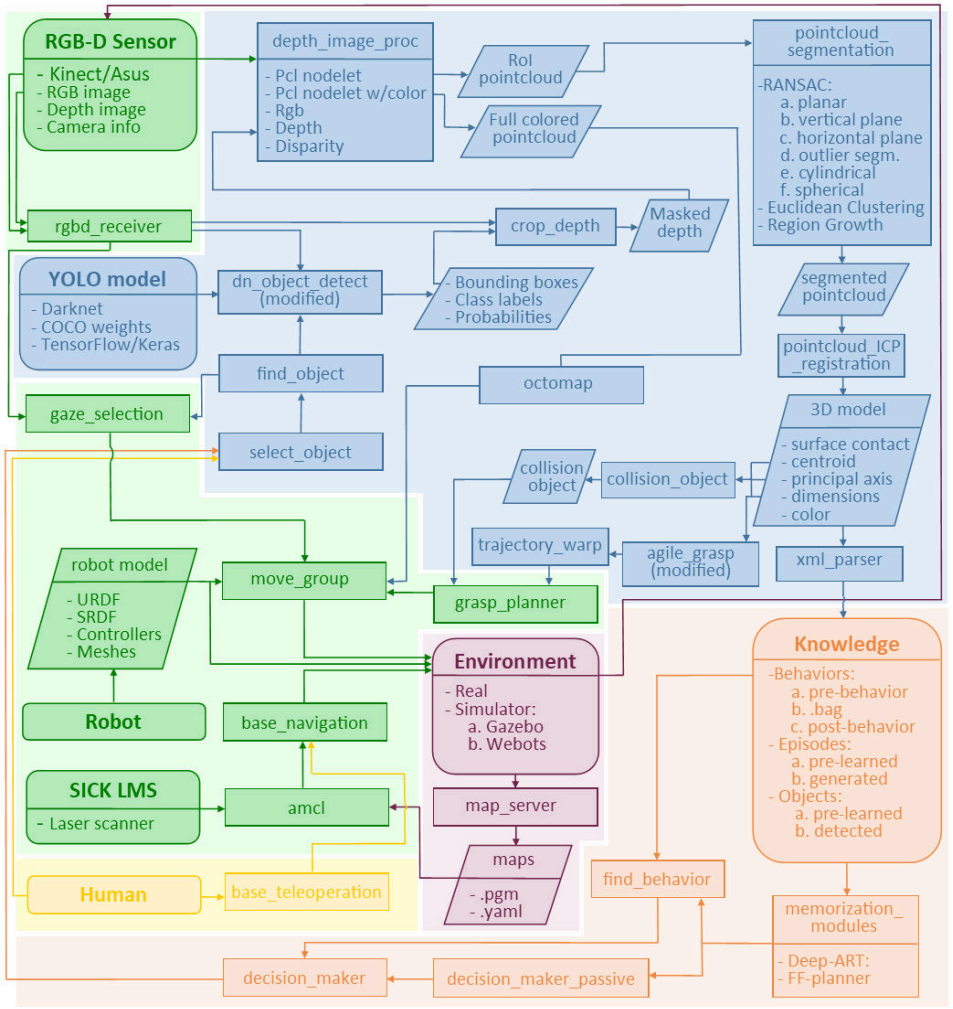


Figure 6.3: Architecture for the solution proposed in **P4**. **Green:** robotic hardware and sensors with their modelling and control, **Blue:** general perception pipeline proposed in this thesis (Section 6.4), **Orange:** memorization modules (Section 6.5), **Purple:** real and simulated environment, **Yellow:** possible, but not necessary, influence of a human user on the system

The Open Motion Planning Library (OMPL) [122], intrinsic to *MoveIt!*, has been used for motion planning, control and collision checking. The usage of planning algorithms such as *RRT** and, later on, *Quick-RRT** [123] allows the robots end-effectors to reach a certain goal by randomly sampling states and generating a collision free trajectory.

Similarly, all the robots sensors are modelled and the relative frames to the robot are defined. This way, the acquired sensory data can be processed correctly and used in the required applications. Common sensors used in this project are laser scanners (SICK LMS) and RGB-D sensors (*Kinect* or *Asus Xtion*) for which ROS packages have been developed for a plug-and-play integration with the ROS framework. The sensory data is required by specific subsystems, some examples are: the Adaptive Monte Carlo localization (*amcl*) package that uses laser scan data for Grid-SLAM. Also, Figure 6.2(c.) displays the robots model in *RViz* with an octomap based on the point cloud obtained by casting into 3D the RGB-D data obtained by an *Xtion* sensor. An example of the flowchart and information exchange between ROS nodes for **P4** is displayed in Figure 6.3.

6.3 Libraries employed

Many libraries and frameworks have been used throughout this project to automate certain routines and to interchange data between subsystems. The majority of the code has been written in *C++* because of its great synergy with all libraries. However, some codes have also been written in *Python* to exploit some of its benefits for certain applications. Additionally, when working with Neural Networks, *Jupyter Notebooks* has been used to develop documents that include live code, equations, visualizations and narrative text. The following are the most relevant libraries:

ROS [4] The Robot Operating System is a set of open-source libraries and tools that help build robotic applications in a wide range of platforms. The true power of this framework is the constant updates by the community that add world-class systems in any, and all, robotic matters. The most relevant examples are: **(a.)** *Rviz*, for powerful 3D data visualization. **(b.)** *RQT*, a Qt-based framework for developing graphical interfaces. **(c.)** *MoveIt!* [125], for the latest advances in motion planning (OMPL [122]), manipulation, 3D perception, kinematics, control and navigation. ROS works similarly to a *master-slave* system where all *nodes* share and receive information from a common *core*.

PCL [126] The PointCloud Library is an set of open-source algorithms for the 3D point cloud processing, essential for robotic perception. Some examples of these algorithms are: feature estimation, surface reconstruction, 3D registration, model fitting, and segmentation.

OpenCV [127] is an open-source computer vision and machine learning software library which includes more than 2500 optimized algorithms, both classical and state-of-the-art. It is used world-wide for real-time vision applications. This library was used extensively in **P1**.

Gazebo [128] The Gazebo simulator is a robust physics engine, with high-quality graphics, and convenient programmatic and graphical interfaces that allows fast algorithm testing, robot prototyping and AI training. A similar system used also in this project is Webots [129].

TensorFlow [124] is an open-source symbolic math library developed by Google for machine learning tasks and specifically for Neural Networks. It has so far become one of the industries standards. Another library used is Keras [130], a high-level API that works on top of TensorFlow allowing for fast and easy NN prototyping.

Others major libraries are: Eigen, Boost, OpenGL and the Python libraries (scipy, numpy, rospy, matplotlib), among others.

6.4 General perception pipeline

Similarly to humans, the perception pipeline is based on the concept that to interact with an object, one first has to find its general position in a cluttered environment. Then one must focus specifically on it to find the necessary geometrical cues for it to be grasped and manipulated. Hence, deep learning methods are applied first to semantically segment classes from their surroundings. Then 3D geometrical segmentations are applied to the RoI to remove noise and, finally, the registration of several views is done to build a correct representation of the object, from which information is extracted that will help the robotic platforms interact with it. An overlook of the framework is shown in Figure 6.4 and 6.5.

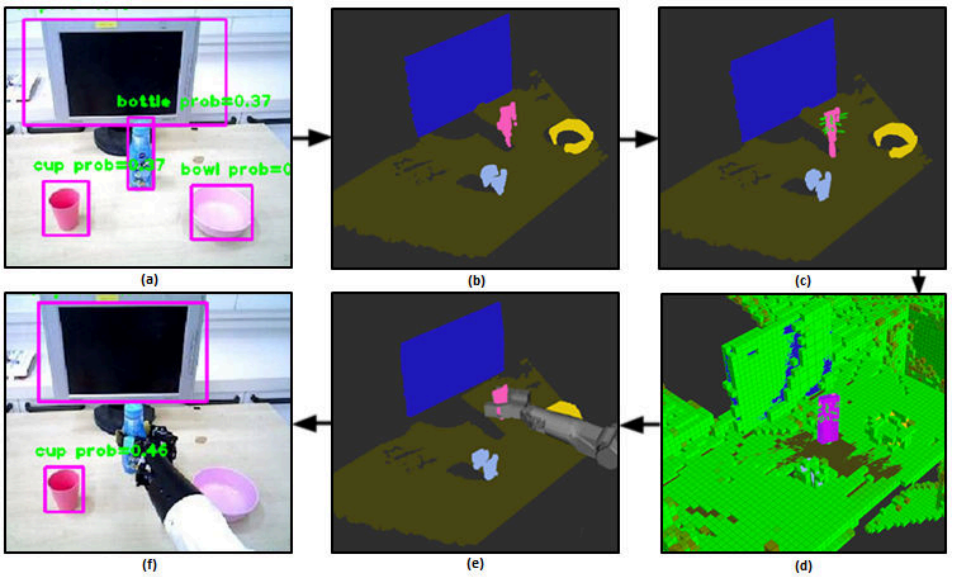


Figure 6.4: End-to-end pipeline of the approach presented in **P3**: **(a.)** Object detection using CNN **(b.)** Segmented objects **(c.)** Array of grasp poses **(d)** Collision object and octomap **(e.)** Planning to target pose **(f.)** Execution of movement.

The following section will detail some key components that were necessary for correctly merging the Deep Learning and 3D segmentation modules onto the robotic platforms. The Robotic Operating System (ROS) was the framework that tied all components together so that all *nodes*/modules were able to exchange information and process it in real time.

6.4.1 Deep Learning modifications

The trained neural networks source code had to be modified to include the ROS libraries which allowed their results to be edited into specific message formats, understandable by all other ROS *nodes*. One example of this is the usage of a modified version of the ROS package *dn_object_detect* [131], which originally displayed the input images with bounding boxes of detected objects layered on top of them, but had to be further modified to generate individual images of bounding box data, class labels and masks. Similarly, the extracted *Mask-RCNN* information had to be converted through bridge *nodes* into specific ROS message types. ROS packages and libraries like *cv_bridge*, *image_transport*, *OpenCV* and *Boost* became essential when constructing these interfacing *nodes*. Likewise, ROS nodes had to be developed that collected low-level data directly from wheel encoders or laser scanners mounted on the robots.

6.4.2 Regions of Interest and Point cloud generation

The collected data was then used to generate individual point clouds for every object instance. The binary masks extracted from the RGB images using DL methods were over-layered on top of the rectified depth images (pixel-to-pixel correspondence) and only the pixels representing the object instances were cast into 3D (Figures 4.2, 4.4 and 6.5), as mentioned in Section 4.1. This is precisely the key for a higher performance and quality increase in the following steps of this perception pipeline: the data has been considerably decimated (RoI) and only the important information is kept. As mentioned, custom ROS message types had to be created so that information was shared between nodes in the most concise and precise was as possible. Some of these messages included the size, position and orientation of bounding boxes, whilst others included the masked RGB image or cropped point cloud with the corresponding label for every object. It is worth mentioning that the computational time required for this process is negligible which entailed that labelled point clouds of all individual classes were generated seemingly in parallel. The update frequency was only dependant on the DNN forward pass speed and in general worked between 20-30 Hz.

6.4.3 3D segmentation and registration

Section 4.2 detailed several geometrical algorithms used to register and segment 3D point clouds. Namely, the *RANSAC*, *Euclidean clustering extraction* and *Region growing segmentation* algorithms were applied to the point cloud object instances. It is worth noting, that the selection of the segmentation method was not randomized. Depending on the objects label, one or another method was used. Some examples of this could be the usage of vertical plane *RANSAC* for TV's, laptops and large kitchen-ware; horizontal plane RANSAC for tables, beds and couches (*inliers*); or for mouse, bowls, cutlery and cellphone (*outliers*) since they are free-form objects typically found on top of

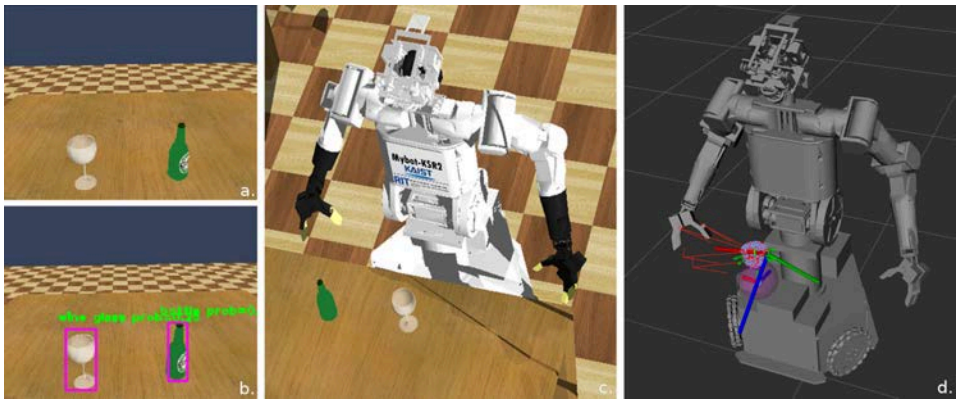


Figure 6.5: Perception of a wineglass in the *Webots* simulator: **(a.)** View from the robot perspective, **(b.)** Segmented RoI using the YOLO architecture, **(c.)** External view of the scene and **(d.)** *Rviz* results showing the full 3D model of the wineglass, its centroid and principal axis, possible grasping poses and the selected best based on the hand (left) closest to the object. A detailed process is seen in Figure 6.6.

planar surfaces; cylindrical *RANSAC* for cups, bottles and vases; spherical *RANSAC* for certain rounded fruits and balls; finally, region growing and euclidean clustering were used for people, animals and objects with a more complex geometry, however their results varied greatly depending on the selected hyper-parameters. It became evident that applying the segmentation algorithms to the RoI point clouds was immediately beneficial, however, the results were prone to errors since these naive geometrical algorithms do not take into account spatial, temporal or semantic knowledge.

A multi-viewpoint model construction was an interesting approach to building full 3D models of the objects whilst, simultaneously, minimizing errors. Since the robotic platforms did not change their position considerably with respect to the objects, it was possible to apply the ICP/NICP algorithms for an accurate point cloud registration (Figure P4.2). If registration failed, it was probably due to poor point cloud generation or large inconsistencies between point clouds at different time steps; therefore the

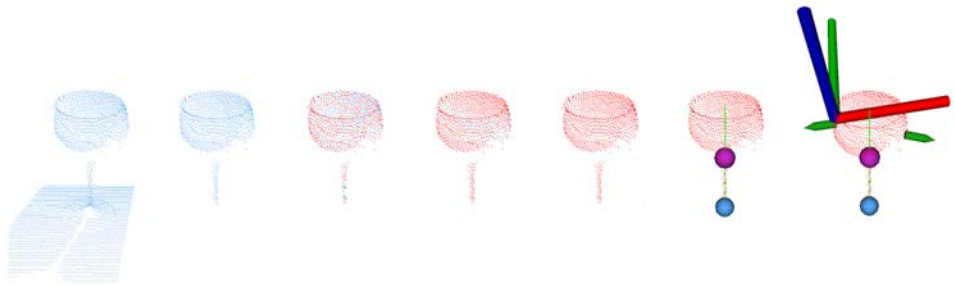


Figure 6.6: **From left to right:** Partially segmented pointcloud obtained from a YOLO mask (Figure 6.5); cylindrical segmentation and registration from different viewpoints; knowledge extraction from the generated model.

registration process was restarted. Once the objects model was fully constructed after several successful registration steps, it was possible to extract information from it, such as position, shape, dimensions, center of mass, orientation, color, etc. This data was fundamental for the posterior interaction.

6.4.4 Collision Object

The physical interaction between robot and objects is desired, but the usage of the *MoveIt!* libraries to achieve it generates a fundamental dilemma: the robot navigation stack is based on the so-called *octomap* of the environment (Figure 6.2.c). That is, a 3D occupancy grid mapping approach that divides the space into available, occupied or unknown voxels. This dynamic method derives the available space and helps plan motions that will not collide with the environment. However, this entails a very complex problem: a robot will never be able to interact with any surrounding objects because to be able to manipulate it, the robot must physically contact the object, which will break the motion planning due to the raised collision in the *octomap*.

To solve this, the collision check between the robot and the object has to be disabled, and to do so, a *collision object* must be created around the object to state that that part of the *octomap* must be treated differently than the rest. Since, generally, the objects full model is not known a priori, the information extracted from the object, by means of the aforementioned perception pipeline (Section 6.4.3), will be used to generate the geometry of the *collision object* (Figure 6.7).

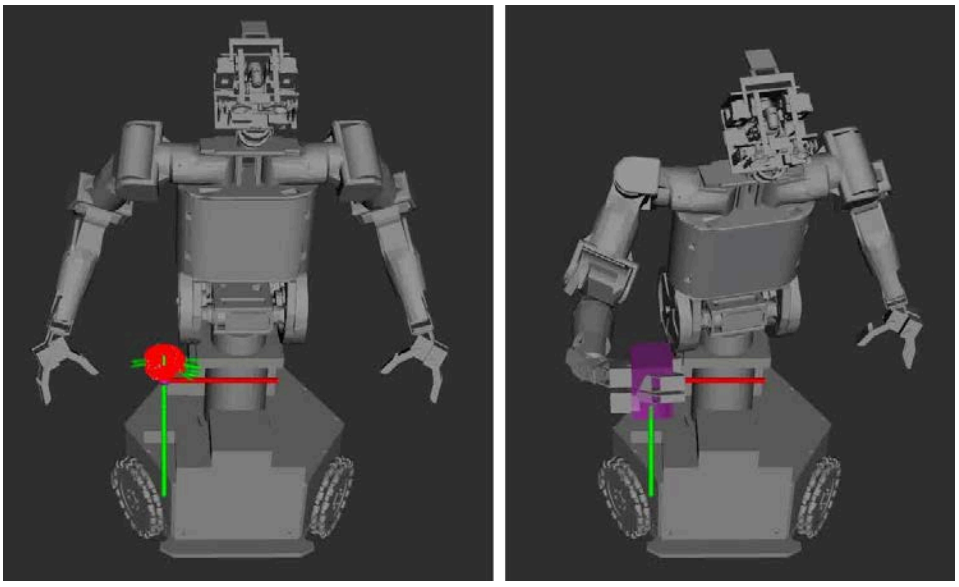


Figure 6.7: *MyBot* humanoid robot semantically segmenting a wineglass, extracting relevant data, inferring grasping poses (green arrows) and generating a *collision object* (purple) to remove collision checking and allow grasping. This figure corresponds to the simulated environment from Figure 6.5. A more detailed description of the process is described in Figure P4.2 from Publication P4.

The *collision object* will be based on three different primitive shapes: cylinders, spheres or rectangular prisms. The final *collision object* could very well be a combination of these, but, for simplicity (less computationally expensive), this option has not been chosen. The selection of primitive form depends on the 3D segmentation method employed during the perception pipeline. The objects extracted information (height, width, depth, radius, principal axis, surface contact point) is then used to generate a 3D oriented *collision object* that overlaps the *octomap* precisely where the objects physical pose is. Therefore, the robot disables the collision checking between itself and that area of the *octomap* corresponding to the object, ultimately allowing the robot to grasp the object.

6.4.5 Grasping

Object grasping is, both, a very complicated and utterly important field for Robot-Object Interaction. Without it, the robot will not be able to correctly interact with the environment. Several approaches to grasping have been researched in the past years: Saxena *et al.* [132] tried to solve this problem by identifying possible grasping points in 2D images using a supervised algorithm that detects visual features. The sparse set of points is then triangulated into 3D to obtain grasping candidates. This approach did not require any 3D input data and could be used for objects outside the training data. However, recent research has focused on training neural networks to achieve correct grasp poses based on RGB images [133], [134], depth images [135], both (RGB-D) [136], [137] or 3D point clouds [138], [139], [140].

Since the 3D model of the object has been derived previously, it would be beneficial to apply a ML algorithm that worked with point clouds to derive correct grasping poses. To this end, the system developed by Pas *et al.* [141], and publicly available as a ROS package (*agile_grasp*), was employed. It requires an input point cloud and the geometric parameters of the robots gripper. To simplify the pipeline and obtain more accurate results, instead of inputting the whole environments point cloud, only the 3D representation of the model is fed into the system. In particular, a large set of grasping hypothesis is sampled based on the objects *antipodal contacts* - line between two points of an object that lies inside the friction cones - and the *antipodal hands* - pose of hand that is not in collision with object but still has at least one pair of *antipodal contacts*. Then, a pre-trained SVM is used to classify the hypothesis correctness and obtain the best possible grasping poses.

The derived grasping poses are evaluated further. The euclidean distance from the objects centroid to the robots arms is calculated to see which arm is closer, simplifying the trajectory planning. The *cosine similarity* θ between all grasping hypothesis orientations (A_i) and an horizontal vector (B_i) is derived. This prioritizes those grasps that are orthogonal to the X-Y plane since most objects can be grasped that way:

$$cos(\theta) = \frac{A \cdot B}{||A_2|| \cdot ||B_2||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{6.1}$$

The trajectory for the selected robot arm to reach the final grasping pose is then computed using *MoveIt!*.

Figure 6.8: Different examples of grasp pose detection and object manipulation. **From left to right:** YOLO RoI, label and probabilities; grasping pose candidates (green arrows); selection of best grasp and execution through *MoveIt!*; robots view of the final grasp.

6.4.6 Trajectory warping

The interaction with objects can only be done after having correctly recognized, segmented and modelled them in 3D. In this project, not only must the robot precisely grasp the objects, but it must also perform certain tasks/actions with them; sometimes

even interaction between more than one object is needed. The behaviours have been pre-learnt through either observing a human action or by moving the robots arm passively [142], stored as *.bag* files of data, using ROS. Additionally, certain behaviors have certain pre- and post-requisites which are detailed in XML files (Algorithm 6.2).

```
<behavior>
    <name> pour </name>
    <type> binary </type>
    <keep_operation> false </keep_operation>

    <pre_behavior>
        <make_collision> <place/> </make_collision>
        <gaze> <place/> </gaze>
    </pre_behavior>

    <main_behavior>
        <arm_type> left </arm_type>
        <check_collision> true </check_collision>
        <compute> true </compute>
        <pose_solve> false </pose_solve>
        <bag_file> bags/left_pour.bag </bag_file>
    </main_behavior>

    <main_behavior>
        <arm_type> right </arm_type>
        <check_collision> true </check_collision>
        <compute> true </compute>
        <pose_solve> false </pose_solve>
        <bag_file> bags/right_pour.bag </bag_file>
    </main_behavior>

    <post_behavior>
        <gaze> <place/> </gaze>
        <remove_collision> <place/> </remove_collision>
    </post_behavior>
</behavior>
```

Algorithm 6.2: Example of a pre-trained behaviour including the steps required before (looking at the objects initial position if its location has already been stored in memory, if not, the robot must find the object in the environment. Then, the collision object has to be generated), during (finding solutions, checking collisions, warping trajectory and if it fails with one arm, try with the other) and after (looking at the final location of the object and removing the generated collision object) the action is done.

However, in a dynamic environment, these behaviours will become obsolete because the clearly defined start and end states will not match the reality. Thus, the trajectory must be warped to accommodate the information from the objects obtained from the perception pipeline [143], **P4**. These behaviours include actions like pouring, throwing and shaking; which rely heavily on the object and end effectors position.

To achieve trajectory warping, first the original trajectory is defined using a twist vector from screw theory: $\xi = [\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T \in I\!R^6$, where ω is an angular vector and v a linear change. Thus, the homogeneous transformation matrix H_s^g between start ($_s$) and goal (^{g}) states is defined as:

$$H_s^g = exp(\hat{\xi}_s^g) = exp\left(\begin{bmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{bmatrix}_s^g \right) \tag{6.2}$$

Similarly, the transformation matrices for trajectories between start and end poses for the demonstrated trajectory and those of the actual environment are, respectively, $H_o^s, H_o^g, \hat{H}_o^s$ and $\hat{H}_o^g$. The final warping variation H_{diff} is:

$$H_{diff} = (H_o^s - H_o^g)^{-1} \cdot (\hat{H}_o^s - \hat{H}_o^g) \tag{6.3}$$

$$\hat{\xi}_{diff} = log(H_{diff}) \tag{6.4}$$

To find the warped transformation matrix $\hat{H}_s^k$ for every *k-th* point, the following equation is applied:

$$\hat{H}_s^k = \hat{H}_o^s H_s^k exp(k \frac{\hat{\xi}_{diff}}{N}) \tag{6.5}$$

where N is the total number of points until the *k-th* point and H_s^k are the end-effectors poses in the original trajectory.

6.5 Memorization and reasoning modules

One of the final components for this project was the memorization element for the Task Intelligence problem, through which a robot is able to learn, reason and execute specific behaviors based on knowledge from the environment. This was accomplished using a combination of Episodic Memory, such as Deep Adaptive Resonance Theory (Deep-ART) [144] developed by the RIT Lab at KAIST and an FF-planner [145].

The Episodic Memory simulates the long-term memory of humans using unsupervised NN. The training cues are defined as combinations of clauses that incorporate information about: the action (v), two objects (o_1 and o_2) and their geometrical relation (γ). They are then encoded into an attribute layer (layer F_1 of Figure 6.9) which are then passed along the network, through Input Field and Input Channel layers, allowing the network to encode *events* based on full or partial cues. The *event* encoding happens in layer F_3. Similarly, the *episodes* (combinations or sequences of *events*) are stored in the F_4 layer. Finally, those episodes that occurred close in time are modelled in layer F_5. When retrieving an episode based on input cues, the network reverses the encoding process, and recalls past sequences of actions that happened in similar situations.

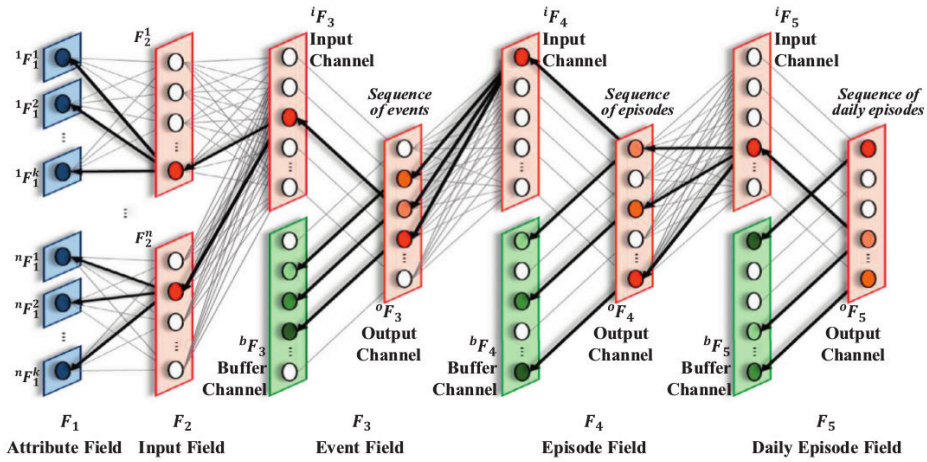


Figure 6.9: Deep-ART architecture

Evidently, Deep-ART can only retrieve sequences of actions it already knows. Since in a dynamic world, many tasks require inferring partial tasks from many episodes, it is necessary to implement a system that can generate unlearned tasks based on similar pre-learned ones: the FF-planner. By using enforced hill climbing (EHC) (or A* algorithm if EHC fails), the subsystem is able to parse into PDDL language [146] the same combinations of clauses as in Deep-ART plus start (s_0) and goal states (s_g). Hence, the FF-planner is able to infer new task sequences to achieve a desired goal state based on pre-learned *events* and *episodes*.

By adding these subsystems onto the perception pipeline, a robot is able to to comprehend and interact with a previously unknown dynamic environment whilst storing all obtained information on the surrounding objects. That is, the robot is able to navigate the environment and remember the positions and characteristics of diverse objects (Algorithm 6.4) which will be later needed to accomplish specific tasks or episodes (Algorithm 6.3). A detailed description of this procedure is shown in [143] and **P4**.

```
<episode>
    <name>bring_wineglass</name>

    <event object="table_a" behaviour="move" place="table_a"/>
    <event object="mybot" behaviour="octo" place="mybot"/>
    <event object="wineglass" behaviour="find" place="wineglass"/>
    <event object="bottle" behaviour="grasp" place="bottle"/>
    <event object="bottle" behaviour="pour" place="wineglass"/>
    <event object="bottle" behaviour="put" place="table_a_place"/>
    <event object="mybot" behaviour="init" place="mybot"/>
    <event object="wineglass" behaviour="grasp" place="wineglas"/>
```

```
    <event object="mybot" behaviour="move" place="table_b"/>
    <event object="wineglass" behaviour="put" place="table_b_place
        "/>
</episode>
```

Algorithm 6.3: Example of the generated episode from Deep-ART and the FF-planner for the task of pouring a glass of wine (P4)

```
<?xml version="1.0" encoding="UTF-8" ?>
<object>
    <name>bottle</name>
    <objtype>dynamic_recog</objtype>
    <real_name>bottle</real_name>
    <visual>
        <color r="0" g="0" b="150" />
    </visual>
    <collision>
        <cylinder>
            <dimension radius="0.0484673" height="0.145268" />
            <pose>
                <position x="0.000208756" y="0.000231245" z="
                    -0.0726331" />
                <rpy r="0" p="0" y="0" />
            </pose>
        </cylinder>
    </collision>
    <behavior>
        <grasp>
            <offset x="-0.0133603" y="0.0248691" z="0.0557853" />
            <quaternion x="0.721747" y="-0.503153" z="0.230283" w="
                -0.415797" />
        </grasp>
        <pour />
        <locate />
        <release />
    </behavior>
</object>
```

Algorithm 6.4: Example of the generated XML object description for the detected wine bottle in P4. The offset in the grasp section is relative to the surface contact point (usually a table).

Chapter 7

Conclusions and Future Research

7.1 Summary of Conclusions

This project has demonstrated that developing an *intelligent* robot that can interact with the world around it is a difficult task. Such a complex system can only be designed based on smaller and more specific subsystems, very much like how the human anatomy works. Many of said modules have been described, developed and used in this thesis: robot model and control, sensor data acquisition and fusion, autonomous navigation, object recognition through diverse Deep Neural Network architectures, 3D point cloud segmentation, object manipulation, trajectory warping, memorization and reasoning. However, the most important developed subsystem is probably the perceptive pipeline since it allows the platform to understand its environment to then be able to act upon it without any prior knowledge. This project has proposed several general perception frameworks, based on the demonstrated synergy between traditional 3D segmentation methods and novel Neural Network architectures, that work for any robotic platform, allowing real time Robot-Object Interaction. To accomplish this, a series of tasks had to be solved first: the contributions of this thesis are described below:

This dissertation explains the different software frameworks and libraries necessary to merge all subsystems. The mastering of the Robot Operating System (*ROS*) is crucial for this endeavour since it can be considered the glue that pastes all modules together in such a complex system. Other employed libraries focus on specific fields like robot modeling, motion planning and control (*MoveIt*), computer vision (*OpenCV*), neural networks (*TensorFlow*), point cloud processing (*PCL*) and physical simulation (*Gazebo/Webots*); all of these elements are of vital importance for the development of a *social/service* robot that can interact with objects.

These tools allow sensor calibration (*Kinect*) and robot modelling, motion planning (autonomous navigation, arm trajectory planning and manipulation), sensory data acquisition, fusion and processing (for localization and perception) and object manipulation. All of these elements have been tested on diverse robotic platforms, from small mobile robots to real-size humanoid robots.

Different Machine Learning approaches have been evaluated for object detection and recognition, mainly (*YOLO*) and (*Mask-RCNN*). They have been trained on available datasets (*COCO, SUN-RGB*) as well as custom ones (*Doors and cabinets, Maritime*), and their models have been enhanced, adding depth as an input, and thoroughly tested on different applications taking into account different performance metrics. These projects include **(a.)** detecting doors and cabinets, finding the handles and opening them, **(b.)** detecting kitchenware and smaller objects on top of tables and grasping them with robotic arms, **(c.)** recognizing, detecting, segmenting and building 3D models of many classes of objects simultaneously: furniture, humans, kitchenware, electronic components... **(d.)** Developing a generic perception pipeline to solve the the Task Intelligence problem.

The architectures that provide the best results have been integrated with the rest of the subsystems in the ROS framework where input images were obtained from the robots sensors, the processing was done in real time and the results were then shared and used by other modules. Their combination with point cloud processing methods has enabled the creation of a generic perception pipeline that simulates generally how humans perceive and interact with their surroundings: a particular object is recognized and detected in a cluttered scene and its general location is established, upon closer inspection, all irrelevant data is discarded and a precise 3D model of the object is registered over time from different viewpoints, finally, some of the objects characteristics are then extracted that help interact with it. The optimization of the perceptive module has allowed for the parallel and real-time semantic segmentation of objects in a dynamic and cluttered environment without any prior knowledge of it; this means the robots are able to build 3D models of the classes they were trained for simultaneously for all objects in a scene. The fusion of 2D and 3D segmentation has proved to yield more accurate and faster results than individual approaches.

The creation of 3D models for all objects through multiple viewpoint registration (*N/ICP*) becomes essential for Robot-Object Learning and Interaction since it allows for 3D feature matching. Additionally, many qualities of the objects (such as form, size, centroid, contact point, principal direction, color and grasping poses) are derived and stored that help the robot interact with or manipulate them. The more accurate the 3D modelling of the object is, the more precise the obtained information will be and, thus, higher probabilities of a successful grasping and manipulation.

The developed general perception pipeline has also been merged with Simultaneous Localization and Mapping (*SLAM*) techniques to add semantic components to it. Therefore, a robot is able to build 3D maps of its surroundings, obtain semantic and physical knowledge of the class instances, navigate autonomously avoiding obstacles and interact with the objects.

Finally, the integration of all established components with an Episodic Memory (*Deep-ART*) module helps solve the Task Intelligence problem. This way, all extracted data for every object instance is memorized, as XML files, and used to accomplished a given general objective. This larger goal is commonly divided into smaller, simpler sub-tasks which can be solved using pre-learned behaviours warped to suit the robots current state and the extracted objects information; or by inferring new sequences suing the FF-planner. Hence, robots are now able to fulfill a certain complex goal, given a partial cue, by solving simpler tasks by inferring or using already learned actions. All the information stored in memory from all segmented objects is used to modify these actions accordingly.

All in all, this project has proved that a highly complex system like a *service* robot is possible, but requires the non-trivial integration and cooperation of many vital components. This project has focused mostly on the perceptive side of such complicated enterprise, but has necessarily touched upon many other subsystems. Thus, a deep analysis was made of what modules were necessary (which could be used off-the-shelf, which had to be modified and which had to be built from scratch), how to put them together and in what order. The grand feat of developing several robotic platforms capable of understanding their surroundings, without any prior knowledge, and then being able to interact with it, with a real time high degree of accuracy and adaptability, has been accomplished.

7.2 Future Research

Even though this project been proven that combining Machine Learning and 3D imaging methods is extremely beneficial for precise Robot-Object interaction, there are a few key points that are worth noting and that, provided further research, could potentially impact greatly the way robots interact with dynamic environments.

The usage of DL techniques as the baseline for a generic perception pipeline does indeed increase performance and accuracy; however, there also exists a limiting factor: the training data sets and detectable classes are, eventually, finite. This means that DNNs can recognize different instances of a subset of trained objects, but will omit many other classes due to them not being previously trained upon. Specifically, the COCO dataset which has been largely used in this project to train DNNs, only has 80 individual classes. To solve this limitation, two approaches could be investigated: **(a.)** The usage of larger and deeper networks that could be trained to recognize and detect a higher number of classes could be a plausible solution. Overfeat [39] was trained on 1.2 million images of up to 1000 different classes. *YOLO9000* [56] can detect over 9000 classes using a similar architecture than its predecessors, which have been used a lot throughout this dissertation. A similar concept is R-FCN-300 [147]. **(b.)** Transfer learning so that a smaller subsample of training data can be generalized to other inputs. Hoffman *et al.* [148] developed a method to annotate the bounding boxes of thousands of categories; however, it required them to be classified beforehand. Likewise, a new method [149], by the same author, was then proposed to leverage weakly supervised data for adapting classifiers to detectors. Recently, transfer learning between different computer vision tasks [150] allowed new annotations to be generated from diverse semantic tasks.

The 3D segmentation of point clouds yielded accurate results; however, the diverse segmentation methods can be considered naive because they only take into account a limited number of geometrical characteristics. Furthermore, the employed segmentation method for every instance was selected by the user based only on the classified label of the object. This approach is prone to errors since these algorithms do not take into account spatial [29], temporal [151] or semantic knowledge. This could be solved by adding an additional reasoning layer on top of the perception pipeline were results from simultaneous segmentation methods were combined to find the optimal object model. Furthermore, loop closure and probabilistic texture mapping for scene and object 3D reconstruction would entail far better results than by just registering point clouds over time.

Finally, the combination of all different subsystems into a robotic platform requires especial attention. The integration was done in a procedural way, meaning that each module had to wait until its predecessor had finished and obtained correct results. Specifically, there was no reasoning mechanisms when one of the subsystems failed, which entailed the propagation of errors to other modules. An inference engine should be added to diagnose faults or detect poor results and act accordingly [152].

Bibliography

[1] K. Schwab, *The Fourth Industrial Revolution*. Crown Publishing Group, 2017.

[2] C. B. Frey and M. A. Osborne, "The future of employment: How susceptible are jobs to computerisation?" *Technological Forecasting and Social Change*, vol. 114, p. 254–280, 2017.

[3] P. Baxter, S. Lemaignan, and J. G. Trafton, "Cognitive architectures for social human-robot interaction," *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016.

[4] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[5] M. Denkowski, "Gpu accelerated 3d object reconstruction," *Procedia Computer Science*, vol. 18, p. 290–298, 2013.

[6] R.-G. Mihalyi, K. Pathak, N. Vaskevicius, T. Fromm, and A. Birk, "Robust 3d object modeling with a low-cost rgbd-sensor and ar-markers for applications with untrained end-users," *Robotics and Autonomous Systems*, vol. 66, p. 1–17, 2015.

[7] J. Xie, Y.-F. Hsu, R. S. Feris, and M.-T. Sun, "Fine registration of 3d point clouds fusing structural and photometric information using an rgb-d camera," *Journal of Visual Communication and Image Representation*, vol. 32, p. 194–204, 2015.

[8] T. Foissotte, O. Stasse, A. Escande, P.-B. Wieber, and A. Kheddar, "A two-steps next-best-view algorithm for autonomous 3d object modeling by a humanoid robot," *2009 IEEE International Conference on Robotics and Automation*, 2009.

[9] M. Jaiswal, J. Xie, and M.-T. Sun, "3d object modeling with a kinect camera," *Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 2014.

[10] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3d object modeling," *The International Journal of Robotics Research*, vol. 30, no. 11, p. 1311–1327, Jul 2011.

[11] A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Autonomous 3d model generation of unknown objects for dual-manipulator humanoid robots," *Robot Intelligence Technology and Applications 5 Advances in Intelligent Systems and Computing*, p. 515–530, 2018.

[12] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[13] M. Fenzi, R. Dragon, L. Leal-Taixé, B. Rosenhahn, and J. Ostermann, "3d object recognition and pose estimation for multiple objects using multi-prioritized ransac and model updating," *Lecture Notes in Computer Science Pattern Recognition*, p. 123–133, 2012.

[14] L. Bo, X. Ren, and D. Fox, "Learning hierarchical sparse features for rgb-(d) object recognition," *The International Journal of Robotics Research*, vol. 33, no. 4, p. 581–599, 2014.

[15] D. Ignakov, G. Liu, and G. Okouneva, "Object segmentation in cluttered and visually complex environments," *Autonomous Robots*, vol. 37, no. 2, p. 111–135, Dec 2013.

[16] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka, "Rigid 3d geometry matching for grasping of known objects in cluttered scenes," *The International Journal of Robotics Research*, vol. 31, no. 4, p. 538–553, Jul 2012.

[17] U. Asif, M. Bennamoun, and F. Sohel, "Real-time pose estimation of rigid objects using rgb-d imagery," *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, 2013.

[18] ——, "Discriminative feature learning for efficient rgb-d object recognition," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[19] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, p. 927–941, 2008.

[20] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using rgb-d cameras," *Lecture Notes in Computer Science RoboCup 2011: Robot Soccer World Cup XV*, p. 306–317, 2012.

[21] C. Goron, L. Z. csaba Marton, G. Lazea, and M. Beetz, "Robustly Segmenting Cylindrical and Box-like Objects in Cluttered Scenes Using Depth Cameras," in *Proc. ROBOTIK 2012, 7th German conference on VDE robotics*, 2013.

[22] A. Llopart, O. Ravn, and N. A. Andersen, "Door and cabinet recognition using convolutional neural nets and real-time method fusion for handle detection and grasping," *3rd International Conference on Control, Automation and Robotics (ICCAR)*, 2017.

[23] B. Pepik, M. Stark, P. Gehler, and B. Schiele, "Teaching 3d geometry to deformable part models," *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[24] Y. Xiang and S. Savarese, "Estimating the aspect layout of object categories," *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[25] A. Richtsfeld, T. Mörwald, J. Prankl, M. Zillich, and M. Vincze, "Learning of perceptual grouping for object segmentation on rgb-d data," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, p. 64–73, 2014.

[26] A. K. Mishra, A. Shrivastava, and Y. Aloimonos, "Segmenting "simple" objects using rgb-d," *2012 IEEE International Conference on Robotics and Automation*, 2012.

[27] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *European Conference on Computer Vision*. Springer, 2012, pp. 746–760.

[28] X. Ren, L. Bo, and D. Fox, "Rgb-(d) scene labeling: Features and algorithms," *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[29] U. Asif, M. Bennamoun, and F. Sohel, "Unsupervised segmentation of unknown objects in complex environments," *Autonomous Robots*, vol. 40, no. 5, p. 805–829, Apr 2015.

[30] A. M. Turing, "I.—computing machinery and intelligence," *Mind*, vol. LIX, no. 236, p. 433–460, 1950.

[31] F. Rosenblatt, "A probabilistic model for visual perception," *Acta Psychologica*, vol. 15, p. 296–297, 1959.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533–536, 1986.

[33] L. Yann, B. Bernhard, D. John S., H. Donnie, H. Richard. E., H. Wayne E., and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *AT&T Bell Laboratories*, 1989.

[34] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[35] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, p. 1527–1554, 2006.

[36] A. Krizhevsky, S. Ilya, and H. Geoffrey E., "Imagenet classification with deep convolutional neural network," *Advances in Neural Information Processing Systems 25*, vol. 18, p. 1097–1105, 2012.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, p. 211–252, Nov 2015.

[38] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, p. 3207–3220, 2010.

[39] P. Sermanet, E. David, Z. Xiang, M. Michael, F. Rob, and L. Yann. (2013) Overfeat: Integrated Recognition, Localization and Detection Using Convolutional Networks. [Online]. Available: https://arxiv.org/abs/1312.6229

[40] K. Simonyan and A. Zisserman. (2014, Sept.) Very Deep Convolutional Networks for Large-Scale Image Recognition. [Online]. Available: http://adsabs.harvard.edu/abs/2014arXiv1409.1556S

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[42] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv e-prints*, Feb. 2015. [Online]. Available: http://adsabs.harvard.edu/abs/2015arXiv150203167I

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *ArXiv e-prints*, Feb. 2016. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160207261S

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, p. 21–37, 2016.

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

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[48] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[49] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *ArXiv e-prints*, Feb. 2016. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160207360I

[50] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[51] R. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[52] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, p. 1137–1149, Jan 2017.

[53] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[54] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, p. 2481–2495, Jan 2017.

[55] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[56] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[57] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *ArXiv e-prints*, Apr. 2018. [Online]. Available: http://adsabs.harvard.edu/abs/2018arXiv180402767R

[58] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. V. D. Laak, B. V. Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, p. 60–88, 2017.

[59] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *ArXiv e-prints*, Sept. 2016. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160903499V

[60] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, S. Sengupta, and M. Shoeybi, "Deep Voice: Real-time Neural Text-to-Speech," *ArXiv e-prints*, Feb. 2017. [Online]. Available: http://adsabs.harvard.edu/abs/2017arXiv170207825A

[61] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, and et al., "Tacotron: Towards end-to-end speech synthesis," *Interspeech 2017*, 2017.

[62] A. Vaswani, S. Noam, P. Niki, U. Jakob, J. Llion, G. Aidan N., K. Aukasz, and P. Illia, "Attention is all you need," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[63] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *arXiv preprint arXiv:1611.02344*, 2016.

[64] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *arXiv preprint arXiv:1705.03122*, 2017.

[65] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[66] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, *et al.*, "Google's multilingual neural machine translation system: enabling zero-shot translation," *arXiv preprint arXiv:1611.04558*, 2016.

[67] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman, "Deep learning for answer sentence selection," *arXiv preprint arXiv:1412.1632*, 2014.

[68] M. Feng, B. Xiang, M. R. Glass, L. Wang, and B. Zhou, "Applying deep learning to answer selection: A study and an open task," *arXiv preprint arXiv:1508.01585*, 2015.

[69] M. Tan, C. d. Santos, B. Xiang, and B. Zhou, "Lstm-based deep learning models for non-factoid answer selection," *arXiv preprint arXiv:1511.04108*, 2015.

[70] Z. Ren, X. Wang, N. Zhang, X. Lv, and L.-J. Li, "Deep reinforcement learning-based image captioning with embedding reward," *arXiv preprint arXiv:1704.03899*, 2017.

[71] L. Zhang, F. Sung, F. Liu, T. Xiang, S. Gong, Y. Yang, and T. M. Hospedales, "Actor-critic sequence training for image captioning," *arXiv preprint arXiv:1706.09601*, 2017.

[72] G. Gkioxari, R. Girshick, P. Dollár, and K. He, "Detecting and recognizing human-object interactions," *arXiv preprint arXiv:1704.07333*, 2017.

[73] B. Dai, S. Fidler, R. Urtasun, and D. Lin, "Towards diverse and natural image descriptions via a conditional gan," *arXiv preprint arXiv:1703.06029*, 2017.

[74] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2425–2433.

[75] H. Noh, P. Hongsuck Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 30–38.

[76] J. Sirignano, A. Sadhwani, and K. Giesecke, "Deep learning for mortgage risk," *arXiv preprint arXiv:1607.02470*, 2016.

[77] L. dos Santos Pinheiro and M. Dras, "Stock market prediction with deep learning: A character-based neural language model for event-based trading," in *Proceedings of the Australasian Language Technology Association Workshop*, 2017, pp. 6–15.

[78] J. Korczak and M. Hemes, "Deep learning for financial time series forecasting in a-trader system," in *2017 IEEE Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2017, pp. 905–912.

[79] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, 2018.

[80] K. Lai, L. Bo, X. Ren, and D. Fox, "Detection-based object labeling in 3d scenes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 1330–1337.

[81] A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Generalized framework for the parallel semantic segmentation of multiple objects and posterior manipulation," *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2017.

[82] ——, "Online Semantic Segmentation and Manipulation of Objects in Task Intelligence for Service Robots," *15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2018.

[83] U. Asif, M. Bennamoun, and F. A. Sohel, "Rgb-d object recognition and grasp detection using hierarchical cascaded forests," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 547–564, 2017.

[84] ——, "A multi-modal, discriminative and spatially invariant cnn for rgb-d object labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 9, p. 2051–2065, Jan 2018.

[85] J. Mccormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[86] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[87] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, p. 1697–1716, 2016.

[88] N. Yoshikatsu, T. Keisuke, T. Federico, and S. Hideo. (2018) Fast and accurate semantic mapping through geometric-based incremental segmentation. [Online]. Available: http://arxiv.org/abs/1803.02784

[89] J. Jeong, T. S. Yoon, and J. B. Park, "Multimodal sensor-based semantic 3d mapping for a large-scale environment," *Expert Systems with Applications*, vol. 105, p. 1–10, 2018.

[90] S. Kowalewski, A. Llopart, and J. C. Andersen, "Semantic mapping and object detection for indoor mobile robots," *International Conference on Robotics and Computer Vision (ICRCV)*, 2018.

[91] M. Labbe and F. Michaud, "Memory management for real-time appearance-based loop closure detection," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[92] ——, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, p. 734–745, 2013.

[93] ——, "Online global loop closure detection for large-scale multi-session graph-based slam," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[94] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[95] O. Faugeras and O. A. FAUGERAS, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[96] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–607.

[97] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[98] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: science and systems*, vol. 2, no. 4, 2009, p. 435.

[99] Y. He, B. Liang, J. Yang, S. Li, and J. He, "An iterative closest points algorithm for registration of 3d laser scanner point clouds with geometric features," *Sensors*, vol. 17, no. 8, p. 1862, 2017.

[100] 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*, vol. 24, no. 6, pp. 381–395, 1981.

[101] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 176–183.

[102] K. P. Murphy, "Bayesian map learning in dynamic environments," in *Advances in Neural Information Processing Systems*, 2000, pp. 1015–1021.

[103] S. Thrun, B. Wolfram, and F. Dieter, *Probabilistic Robotics*. The MIT Press, 2006.

[104] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[105] S. Yang, J. Wang, G. Wang, X. Hu, M. Zhou, and Q. Liao, "Robust rgb-d slam in dynamic environment using faster r-cnn," in *3rd IEEE International Conference on Computer and Communications (ICCC)*, 2017, pp. 2398–2402.

[106] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[107] R. Scona, M. Jaimez, Y. R. Petillot, M. Fallon, and D. Cremers, "Staticfusion: Background reconstruction for dense rgb-d slam in dynamic environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[108] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[109] D. Nister, "An efficient solution to the five-point relative pose problem," in *Proc. IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2003, pp. II–195.

[110] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, p. 133, 1981.

[111] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, November 2007.

[112] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*. Springer, 2014, pp. 834–849.

[113] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[114] M. A. Nielsen, *Neural networks and deep learning*. Determination press USA, 2015, vol. 25.

[115] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[116] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[117] 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*. Springer, 2014, pp. 740–755.

[118] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 567–576.

[119] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: a database and web-based tool for image annotation," *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[120] A. B. Beck, N. A. Andersen, J. C. Andersen, and O. Ravn, "Mobotware–a plug-in based framework for mobile robots," *IFAC Proceedings Volumes*, vol. 43, no. 16, pp. 127–132, 2010.

[121] C. A. Witting, "Laser based navigation using ros and guidebot," Bachelor's Thesis, Technical University of Denmark, June 2016.

[122] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[123] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Rrt*-quick: A motion planning algorithm with faster convergence rate," in *Robot Intelligence Technology and Applications 3*. Springer, 2015, pp. 67–76.

[124] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[125] S. Chitta, I. Sucan, and S. Cousins, "Moveit ros topics," *IEEE Robotics and Automation Magazine*, vol. 19, pp. 18–19, 2012.

[126] 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.

[127] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[128] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." in *IROS*, vol. 4. Citeseer, 2004, pp. 2149–2154.

[129] Webots, "http://www.cyberbotics.com," commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[130] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[131] X. Wang, "dn_object_detect," https://github.com//kunle12/dn_object_detect, 2016.

[132] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[133] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3406–3413.

[134] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[135] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *arXiv preprint arXiv:1804.05172*, 2018.

[136] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[137] U. Asif, J. Tang, and S. Harrer, "Graspnet: An efficient convolutional neural network for real-time grasp detection for low-powered devices." in *IJCAI*, 2018, pp. 4875–4882.

[138] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1957–1964.

[139] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *arXiv preprint arXiv:1703.09312*, 2017.

[140] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning," *arXiv preprint arXiv:1709.06670*, 2017.

[141] A. ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," in *Robotics Research*. Springer, 2018, pp. 307–324.

[142] A. Jain, S. Sharma, T. Joachims, and A. Saxena, "Learning preferences for manipulation tasks from online coactive feedback," *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1296–1313, 2015.

[143] D.-H. Kim, G.-M. Park, Y.-H. Yoo, S.-J. Ryu, I.-B. Jeong, and J.-H. Kim, "Realization of task intelligence for service robots in an unstructured environment," *Annual Reviews in Control*, 2017.

[144] G.-M. Park, Y.-H. Yoo, D.-H. Kim, and J.-H. Kim, "Deep art neural model for biologically inspired episodic memory and its application to task performance of robots," *IEEE transactions on cybernetics*, 2017.

[145] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[146] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 639–646.

[147] B. Singh, H. Li, A. Sharma, and L. S. Davis, "R-fcn-3000 at 30fps: Decoupling detection and classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1081–1090.

[148] J. Hoffman, S. Guadarrama, E. S. Tzeng, R. Hu, J. Donahue, R. Girshick, T. Darrell, and K. Saenko, "Lsda: Large scale detection through adaptation," in *Advances in Neural Information Processing Systems*, 2014, pp. 3536–3544.

[149] J. Hoffman, D. Pathak, E. Tzeng, J. Long, S. Guadarrama, T. Darrell, and K. Saenko, "Large scale visual recognition through adaptation using joint representation and multiple instance learning," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 4954–4984, 2016.

[150] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3712–3722.

[151] D. Held, D. Guillory, B. Rebsamen, S. Thrun, and S. Savarese, "A probabilistic framework for real-time 3d segmentation using spatial, temporal, and semantic cues." in *Robotics: Science and Systems*, 2016.

[152] M. C. Capolei, A. Llopart, H. Wu, S. Tolu, and O. Ravn, "A rule-based approach for constrained motion control of a teleoperated robot arm in a dynamic environment," in *Proceedings of the International Conference on Robotics Systems and Automation Engineering (RSAE)*, 2018.

Publication P1

Door and Cabinet Recognition Using Convolutional Neural Nets and Real-time Method Fusion for Handle Detection and Grasping

Adrian Llopart[1], Ole Ravn[1], Nils A. Andersen[1]

Abstract

In this paper we present a new method that robustly identifies doors, cabinets and their respective handles, with special emphasis on extracting useful features from handles to be then manipulated. The novelty of this system relies on the combination of a Convolutional Neural Net (CNN), as a form of reducing the search space, several methods to extract point cloud data and a mobile robot to interact with the objects. The framework consists of the following components: The implementation of a CNN to extract a Region of Interest (ROI) from an image corresponding to a door or cabinet. Several vision based techniques to detect handles inside the ROI and its 3D positioning. A complementary plane segmentation method to differentiate door/cabinet from the handle. An algorithm to fuse both approaches robustly and extract essential information from the handle for robotic grasping (i.e. handle point cloud, door plane model, grasping locations, turning orientation, orthogonal vector to door). A mobile robot for grasping the handle. The system assumes no prior knowledge of the environment

Keywords: convolutional neural network, image processing, point cloud processing, mobile robot, door recognition

1 Introduction

With the technological improvements of the last years, mobile robots have become greatly autonomous, capable of fulfilling diverse services and tasks without human intervention. Despite this, further research can still be conducted to achieve full autonomy in the exploration of human-made environments.

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **(adllo, or, naa)@elektro.dtu.dk**

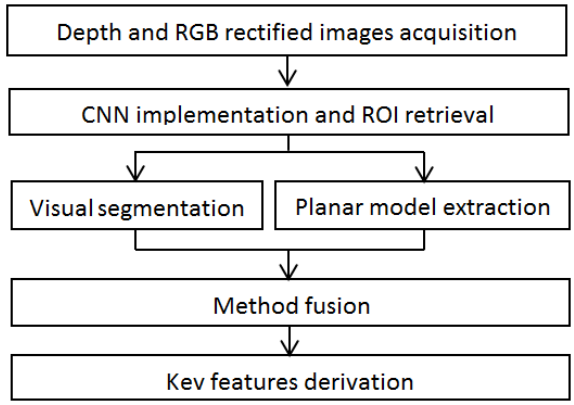


Figure P1.1: Pipeline of the proposed method

A very clear example of this is the necessity of human intervention to open closed doors when robots move in an indoors environment. Several approaches have been explored to solve this situation; however, most of them have focused only on the detection of doors and not interacting with them. These techniques incorporate the usage of either images ([1], [2] and [3]) or depth data ([4] and [5]). Some approaches do indeed deal with the manipulation of door handles, but also present some limitations: [6] requires a prior semantic map; [7] directly detects handles using a 2D sliding window classifier but assumes the robot has already detected a door and is right in front of it. [8] can easily be considered state-of-the-art due to its methods of detecting doors and handles and manipulating them; however the generation of the necessary data takes 10 seconds plus some extra computation time to finally be able to open a door. The pipeline presented in this paper will challenge these limitations.

With the development of pattern recognition and artificial intelligence techniques, novel algorithms are proposed that, in a manner, replicate the learning process and adaptability of human beings to unknown and dynamically changing scenario whilst, simultaneously, reducing computational time. This paper presents a framework that takes advantage of a CNN (similarly to the one proposed in [9]) to generate a ROI, which will significantly reduce computational time, and false positive rates in later stages of the system. Thus, the main contribution is a real-time integrated approach to the detection and manipulation of doors and cabinets, with no prior knowledge of the environment.

To achieve this, rectified RGB and Depth images will be obtained from a Kinect sensor. A Convolutional Neural Net, previously trained over several hundred door and cabinet images from *ImageNet* [10], will take the rectified RGB image as input and generate bounding boxes around doors and cabinets. Two methods will then be used to obtain the handles point cloud: The first one is a visual segmentation approach based on k-means color clusterization of the region of interest. The second one is a plane model extraction of the point cloud generated inside the ROI. Then, the results from both techniques will be merged and a final estimate of the handles point cloud will be produced, from which essential key features will be derived. Finally, a mobile platform will grasp the handle. The entire process is shown in Figure P1.1.

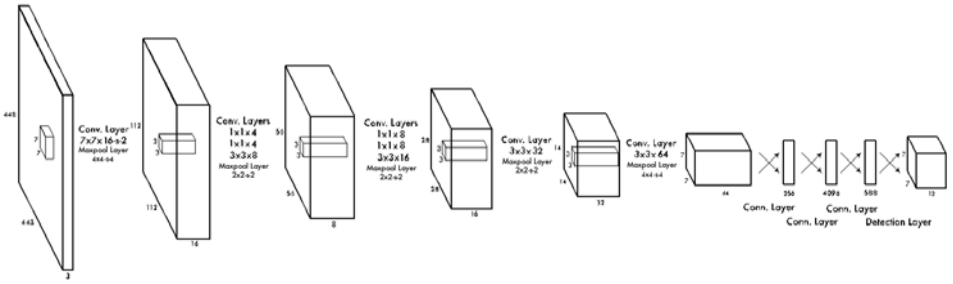


Figure P1.2: Architecture of the Convolutional Neural Net. The system models detection as a regression problem to a 7x7x12 tensor. This tensor encodes bounding boxes and class probabilities for all objects in the image.

2 Convolutional Neural Net

The initial step of the presented method is the use of CNN as a preliminary approach to recognize, detect and segment a ROI out of a full image. The vital use of a CNN will increase responsiveness and performance since it will significantly reduce the amount of data that needs to be processed in later steps of this framework. To achieve this, the proposed CNN must be extremely fast in its entire process, optimized and reliable, generating correct bounding boxes with a high level of precision around those objects it has been trained to detect. Hence, a unified architecture that predicts bounding boxes and class probabilities directly from full images is required: The YOLO Detection System [11]. Since real time processing of video streams with *state-of-the-art* performance is of utmost importance, a small neural net model is applied in the proposed methodology: the *Darknet Reference Model*. The speed of execution of the neural net is directly proportional to the GPU utilized. With a GeForce GT 730 and 2 classes, the Net reaches 14 FPS. These two classes are doors and cabinets (the latter also includes drawers and lockers). The images are selected from *ImageNet* [10], with a total of 510 images for the doors, and 420 for the cabinets. These images were resized to a maximum of 448 pixels. The training took 3 days considering the learning ratios and hardware. Some results of the bounding boxes found by the CNN can be seen in Figure P1.3. Lastly, a great advantage of using a CNN at the beginning of the process is that the system can be expanded by training it to detect other objects and interact with them accordingly, without affecting the performance of door and handle detection presented in this paper.

(a) Cabinet n°1 (b) Cabinet n°2 (c) Red door (d) Cabinet n°3

Figure P1.3: Resulting bounding boxes for several cabinets and doors. The resulting door and handle point clouds are depicted in Fig. P1.8

3 Handle's point cloud generation

As aforementioned, the execution of a trained neural net results in the selection of a ROI around the identified object. All information outside the bounding box is neglected from further processing, ensuing faster and more precise results. Two approaches will be considered to derive the final handle point cloud: visual segmentation and planar model extraction, which will later be fused for optimal results.

3.1 Method 1. Visual segmentation

Two assumptions are taken into consideration when visually detecting handles and generating the corresponding point cloud. The former, the region of interest given by the CNN correctly includes only the desired object (door/cabinet surface). The latter, door and cabinet surfaces have a clear color contrast with their handles. If both of these premises are fulfilled, a k-means color clustering can be applied to the ROI.

With this method, the image is partitioned into k clusters (in this case specifically, and given the second assumption, $k = 2$). The implementation of this method to the given ROI, outputs an image containing only two colors; ideally the one corresponding to the surface of the door and the one of the handle. The goal is to create a clear differentiation between door plane and handle, from which a mask can be extracted. One of the main false positives that this method may generate, is the detection of the lock beneath the handle. To limit the detection specifically to handles, the contours (obtained by applying a Canny Edge Detector) are evaluated in size, form and orientation. The outcome of the ROI extraction, k-means color clustering and contour segmentation is a binary mask which robustly showcases the handle. The whole process is depicted in Figure P1.4.

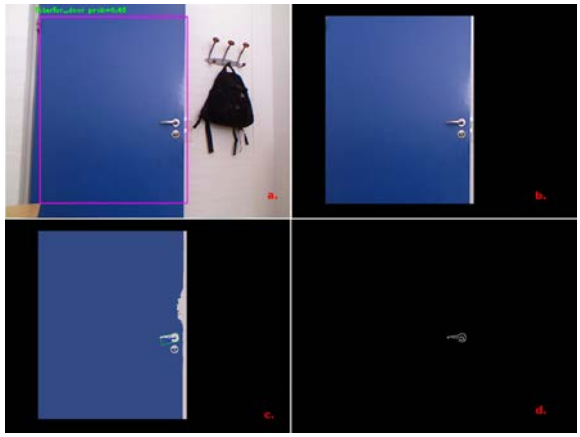


Figure P1.4: **Method 1: Visual segmentation**. **a.** ROI selection from the output of the CNN (with class name and probability written on top). **b.** Cropped rectified RGB image. **c.** K-means clustering of ROI with rotated rectangles (green) and centers (blue) drawn over those contours that fit into certain specifications. **d.** Final binary mask showing only the contour corresponding to the handle, to be applied later to the depth image to generate the handles point cloud.

To obtain the handles point cloud, a rectified depth image is required which is obtained directly form a Kinect sensor and the ROS Indigo interface. It has been transformed and projected onto the same plane as the rectified RGB image (used previously on the Neural Net and the visual segmentation) and is sufficient and necessary to generate a point cloud. Thus, the resulting binary mask from the visual segmentation is applied to the rectified depth image and knowing the intrinsec parameters of the camera, the handles point cloud can be generated.

The derived point cloud will display the door handles 3D information precisely but might incorporate some impurities. In the case of doors, this noise is partly removed by cropping the point cloud around the range of 80 to 120 cm in height (because that is where handles usually are). Even though it is uncommon, some other elements might have survived the process. These will be removed in a posterior process when the handle point cloud obtained by the described visual segmentation method and the one generated from plane extraction are merged together. This is explained in section III-C.

3.2 Method 2. Planar model extraction

As seen in the previous section, a point cloud can be easily generated given a depth image. In the preceding method, the door handle was extracted through segmentation in the RGB space to create a mask which was then applied to the depth information to obtain the point cloud. In this new method, however, the full point cloud of the environment is generated from the original ROI, given by the neural net. Then, a plane model segmentation is applied to it.

This method focuses primary on point cloud manipulation via the PCL library [12] as a secondary way of extracting the handles point cloud. To apply the following planar extraction, the full environments point cloud could be used, as done in [8], but that would use more information than necessary and definitely slower the processing time.

Hence, to allow real time processing, the point cloud needs to be reduced. To solve this issue, the initial binary mask from the ROI given by the CNN is applied to the original depth map so that the generated point cloud only incorporates points inside the ROI and, thus, only reflects the door and handle themselves. The number of points (resolution) of the selected region is kept the same but everything else is deleted, meaning faster processing time. Once again, a pass through filter is applied to remove all those points out of 80-120 cm height region in which handles are usually found for doors.

The strength of this approach resides in the planar segmentation of the point cloud, that is, finding all those points that support a plane model. If the ROI provided by the CNN is of a high degree of precision, the generated point cloud will only be that of the door. Therefore, a planar model, that is supported by the vast majority of points, is extracted from the point cloud, which will represent the door surface. Those points that fit into the plane model given a certain threshold (3-4 centimeters) will be considered as inliers; and those that fail to follow the model will be outliers. Being able to obtain a model of the door allows for robots to navigate closer to the door if they are too far away to detect the handle. This will be necessary since, as shown in the experimental results (V), the door handles become harder to detect after the 1.5 meter distance mark.

This method does sometimes provide false positives: small parts of the neighboring walls or the environment behind a semi-open door might also represented, but their removal will be dealt with, once again, in the method fusing process.

3.3 Fusing both methods

The problem with the first method (visual segmentation) is that items located on top or around the door plane might be detected as false positives when applying the k-means clustering and will appear in the resulting point cloud. The main issue with the second method (planar model extraction) is that all points that do not fit into the plane model are kept in the point cloud. Hence, even though both methods obtain accurate and robust results, for specific cases, false positives are generated and must be dealt with.

A solution that removes false positives whilst keeping the accuracy of both methods is easy to implement. Similar to a bitwise AND operator where the output of the operation is true if, and only if, both inputs are also true simultaneously.

The application of a similar approach to all points in both point clouds results in a highly precise detection of only handles. Therefore, a comparison is done to see if a certain point exists simultaneously in the point clouds generated by the two methods. With this simple process, the imperfections of each method are overcome. All the objects (like posters, signs or even locks) clustered by the visual method will be removed because those points will be considered as inliers in the second method. Likewise, the possible errors of the second method will be removed because they will not appear in the resulting point cloud of the first. An evident example of this is when the ROI is not extremely precise and parts of the environment behind or around the door will show up inside the ROI. Even though, the second method will output them as true since they are outliers of the plane model, the first (visual) method will reject them. This is because the contour of those parts will be too large and certainly too vertical to be able to be recognized as handles, as shown in Figure P1.5. The outcome of merging both methods is a consistent, precise and robust estimation of the handles point cloud. From this result, several handle features can be obtained.

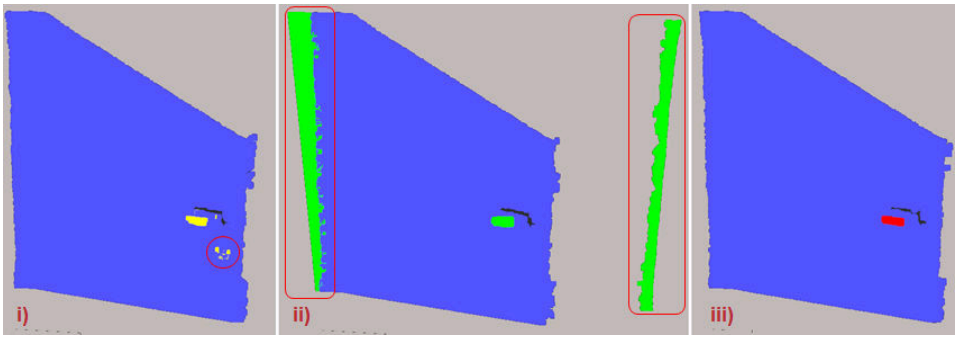


Figure P1.5: **Method fusion.** **i)** Handle point cloud (yellow) obtained from the visual segmentation method (lock is included in the result, which is not desirable). **ii)** Handle point cloud (green) derived from the planar model extraction method (walls are included in the result but need to be removed). **iii)** Final handle point cloud (red) after merging both methods (all errors from each method have been solved).

3.4 Extraction of key data from final door handle point cloud

Once the final handle point cloud is derived, some key features need to be determined (as presented in [7]) to allow robotic manipulation. These key features are: the orthogonal vector to the door plane, the turning direction of the handle, the point around which the handle turns and a possible grasping position. Since the coefficients of the door/cabinet plane are known, the orthogonal vector to the surface can be derived.

The handle turn orientation (either clockwise or anticlockwise) is determined by knowing if the handle is closer to the right edge of the door (thus the turn needs to be anticlockwise) or to the left (clockwise). To achieve this, the centre of the door handles contour is found and the distances to both edges of the ROI are calculated to see which one is smaller. This is done during the visual segmentation method.

The handles turning point will be the edge point in the Y plane of the point cloud, in one or the other direction, depending on the handles turn orientation (clockwise means said point is on the right side of the point cloud and viceversa). The possible candidates for these turning points are shown in green and yellow in Figure P1.6, P1.7a and P1.7b.

Finally, the grasping position is left to the robots interpretation. The centroid of the point cloud and the principal direction are derived. All points are then projected onto said vector resulting in a handle axis with all points from the point cloud projected onto it. Thus, the robot is able to infer the location of grasp by choosing any of the points.

Figure P1.6: The handle (red) and door plane (clear blue) point clouds obtained after using both methods are layered on top of the full environments point cloud. This visual segmentation method for this case is seen in Fig. P1.4 and the method fusion corresponds to Fig. P1.5. The green and yellow points on the left and right side, respectively, of the handle point cloud are some of the key features required for future robotic manipulation.

Figure P1.7: Mobile robot grasping door and cabinet (drawer) handles

The centroid is, however, highly recommended, but there are other possibilities too.

4 Mobile robot and manipulator

A mobile platform (Figure P1.7) has been developed to put into practice the methods presented in this paper. It has been built as a combination of an *iRobot-ATRV-Jr*, an *UR5* robotic arm, a *Robotiq 2-Finger Adaptive Robot Gripper*, a *Robotiq Force-torque* sensor and a *Kinect* camera. The CNN mentioned before runs on a GeForce GT 730M GPU installed on an external computer that communicates directly with the robot. All components have been connected and share information via the ROS *Indigo* interface over Linux 14.04 (LTS) Trusty Tahr distribution.

With the use of the *MoveIt* library [13], the manipulator achieves the grasping position (the centroid of the handle) and orientation, avoiding self collision and other obstacles. In practice, it was seen that the best grasping results were achieved when the goal position was set to be a few centimeters further away from the surface following the orthogonal vector to the door. Then the manipulator can slowly approach the handle following that same vector until the force sensor detects a contact with the door surface. Finally, closing the *Robotiq* gripper ensures a correct grabbing of the handle (Figure P1.7). Additionally, if the detected handle is out of reach, the robot is able to navigate closer to the door first and then attempt the handle grasping.

5 The experimental results

The hardware used for the door recognition via CNN is as follows: Intel Core i7-6700 CPU @ 3.40GHz with 8Gb of RAM and an NVIDIA GeForce GT 730 GPU with CUDA. The performance and speed of the neural net can be improved with a more advanced

Table P1.1: Door detection

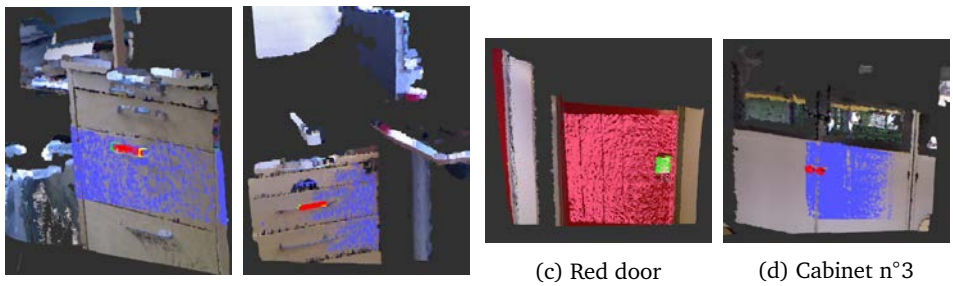
	Shalaby *et al.* [2]			Yang *et al.* [3]			Proposed CNN		
	N_d	TP	FP	N_d	TP	FP	N_d	TP	FP
Simple	55	100%	3.6%	55	98.2%	1.8%	63	98.4%	7.53%
Medium	94	82.7%	17.3%	93	91.4%	1.1%	107	95.3%	4.7%
Complex	63	68.2%	31.7%	56	85.7%	7.1%	71	74.6%	8.45%
Total	212	82.8%	26.4%	204	91.7%	2.9%	241	90.0%	6.64%

setup. For the training step, a batch size of 64, a momentum of 0.9 and a decay of 0.0005 were used.

The CNN proposed in this paper was tested on 210 images (241 doors and cabinets [N d]) and achieved a true positive detection rate (TP) of 90.0% and a false positive rate (FP) of 6.64%. Comparatively, the proposed CNN has a higher detection rate of doors and cabinets than [2], [4], [5] and [9], but marginally lower than [3].

Most of these erroneous results (FP) occur when blank walls next to the objects are considered positive. These results do not generally affect the posterior handle detection rates though, since usually no handles exist in that region. However, to safeguard completely against these errors, those detections with a lower confidence than a certain threshold are completely removed. This is done because it is better to have a higher omission rate than a FP rate, especially considering that the proposed CNN has an update rate of 14Hz and so new images are processed constantly.

The proposed system was tested for accuracy and consistency on door handles. To do so, several door and robot positions (Table P1.1) were evaluated (similarly to [4]): the robot was located in front of a closed door (0°), a semi- open door (35°) and open door (70°), at ranges of 0.5, 1 and 1.5 meters. The same experiment was repeated



(a) Cabinet n°1 (b) Cabinet n°2 (c) Red door (d) Cabinet n°3

Figure P1.8: Each image displays the corresponding handle point cloud from the ROIs extracted in Fig. P1.3. The handle point clouds are represented in red and the detected planar surface in blue (except for the case of the red door in which it is green and pink respectively). it is worth noticing how, with some simple minor tweaks to the algorithm (like what is the minimal distance between points for them to be considered as part of the same plane or size and orientation of the rotated rectangles around the segmented handle clusters), it can be easily adapted to non rectangular handles, as shown in the *Red door* and *Cabinet n°3* images.

for these three distance ranges but with a relative angle between door normal and robot of 30 degrees on both sides. To test the performance of the algorithm, each test was evaluated during 10 seconds and the number of attempts that provided a good, unknown or bad result were recorded. A good result is considered when the obtained handles point cloud is at least as big as half of its surface. An unknown result is an attempt that fails to provide any type of point cloud. A bad result is a point cloud that does not strictly correspond to the handle. The full outcome is presented in Table P1.2.

It is evident that for ranges going up to 1.5 meters, the presented method is able to detect accurately door planes and its handle point cloud. As seen in the results, the further you move away from the door, the worse the performance is. This is mainly due to the following factors:

- Firstly, the small errors during the calibration process become more evident the further away you are from the door; this means that when rectifying depth and RGB images, the pixel-to-pixel matching will not be perfect, thus, generating a point cloud that is not correct. This is easily seen around the handle area, where pixels that have been successfully detected as handle are being represented on the point cloud as part of the door plane. For larger ranges than 1 meter, this is one of the reasons for attempts resulting in *unknowns*.

- The further away the robot is from the door, the looser the ROI that the CNN outputs generally is, meaning some parts of the surrounding image might be included inside it. The consequence is that some parts from the wall might be presented as belonging to the handles point cloud, thus resulting in an attempt classified as bad. The effects of this problem usually happen after the 1.5 meter barrier.

- Lastly, around the 1.5 meter mark the hardware limitations of the sensor result in an evident increase in the number of unknown because the second method is not able to produce correct handle point clouds. Meeussen et al. [8] presented a system that is able of detecting handles with specularity.

The presented system achieves an average 0.96% error rate at handle detection for a distance of 0.5 meters, 3.11% for 1 meter and 10.55% for 1.5 meters. This clearly outperforms previous approaches at distances shorter than 1.5 meters. Kim et al. [1] only detected 50% of the tested handles; Meeussen *et al.* [8] obtained around a 60% success rate in handle detection; Ruehr *et al.* [6] reached a stunning 93.3% of

Table P1.2: Handle Detection

		CLOSED DOOR (0°)				SEMI-OPEN DOOR (35°)				OPEN DOOR (70°)			
Range (m)	**Angle (°)**	**Good (%)**	**Unknown(%)**	**Bad (%)**	**Attempts**	**Good (%)**	**Unknown (%)**	**Bad (%)**	**Attempts**	**Good (%)**	Unknown (%)	**Bad (%)**	**Attempts**
0.5 m	-30°	70.6	23.5	5.9	17	66.7	28.6	4.8	21	$*^1$	$*^1$	$*^1$	-
	0°	90.5	9.5	0.0	21	100.0	0.0	0.0	16	100.0	0.0	0.0	16
	30°	81.0	19.0	0.0	21	69.2	30.8	0.0	26	68.2	31.8	0.0	22
1 m	-30°	88.9	11.1	0.0	18	93.8	6.2	0.0	16	$*^1$	$*^1$	$*^1$	-
	0°	57.9	36.8	5.3	19	87.5	12.5	0.0	16	61.9	23.8	14.3	21
	30°	55.6	44.4	0.0	18	84.2	10.5	5.3	19	80.0	20.0	0.0	20
1.5 m	-30°	18.8	81.2	0.0	16	18.8	62.4	18.8	16	$*^1$	$*^1$	$*^1$	-
	0°	33.3	60.0	6.7	15	41.2	47.1	11.7	17	14.3	71.4	14.3	14
	30°	11.8	76.4	11.8	17	30.0	60.0	10.0	20	22.2	66.7	11.1	18

$*^1$ In this position, the robot would be looking almost perpendicular to the door plane so the handle extraction was not possible.

true positives using only depth data but had prior knowledge of the door and cabinet locations; Klingbeil *et al.* [7] also reached 93.2% using only visual features. Hence, it is proven that fusing both approaches (visual segmentation and planar model extraction) allows for much better results than each individual method.

The average omission (unknown) rates are 14.65%, 20.66% and 65.65%, respectively. These are not very relevant because the fast update of data ensures results are obtained almost every time step. The average processing time of the end-to-end system (except the grasping) is of 1.8Hz; which is much faster than [5], since that approach does not detect handles, and [8], where just the data collection for one trial works at 1Hz.

It was observed that for all cases in which an object is detected via the neural net, a robust and precise model of its surface is obtained. In cases where the robot is far from the door/cabinet surface, the handle detection precision will be lower. This is not significant because for all cases, the robot will obtain an estimated position of the object and can easily move closer to it for a more definite detection, if required. Lastly, the algorithm was also evaluated in front of diverse types of drawers, cabinets and doors. Some of the results are shown in Figure P1.8.

6 Conclusions and Future Work

The proposed approach accurately recognizes doors and cabinets in dynamically changing environments by virtue of a trained Convolutional Neural Net. The state-of-the-art is pushed further by using a combination of a CNN, two different methods (visual segmentation and planar model extraction) and their fusion to produce a precise and consistent point cloud of the handles. Specific key features can then be extracted so that the designed robot can grasp the handles and act accordingly.

In the future, faster and more precise algorithms will be implemented that detect handles even if they have specularity. Finally, a force feedback control system will be designed to be able to not only grasp handles more securely, but also to, ultimately, open doors and cabinets.

Bibliography

[1] S. Kim, H. Cheong, D. H. Kim, and S. K. Park, "Context-based Object Recognition for Door Detection," in *Proc. IEEE International Conference on Advanced Robotics: New Boundaries for Robotics (ICAR)*, Tallinn, Estonia, Jun. 2011, p. 155–160.

[2] M. M. Shalaby, M. A. M. Salem, A. Khamis, and F. Melgani, "Geometric Model for Vision-based Door Detection," in *Proc. IEEE International Conference on Computer Engineering and Systems (ICCES)*, Kuala Lumpur, Malaysia, Sep. 2014, p. 41–46.

[3] X. Yang and Y. Tian, "Robust Door Detection in Unfamiliar Environments by Combining Edge and Corner Features," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops (CVPR Workshops)*, San Francisco, California, USA, Jun. 2010, p. 57–64.

[4] T. H. Yuan, F. Hashim, W. Zaki, and A. B. Huddin, "An Automated 3D Scanning Algorithm using Depth Cameras for Door Detection," in *Proc. Electronics Symposium: Emerging Technology in Electronic and Information*, 2015, p. 58–61.

[5] S. M. Borgsen, M. Schoepfer, L. Ziegler, and S. S. Wachsmuth, "Automated Door Detection with a 3D-Sensor," in *Proc. Canadian Conference on Computer and Robot Vision (CRV)*, Montréal, Quebec, May 2014, p. 276–282.

[6] T. Ruehr, J. Sturm, and D. Pangercic, "A Generalized Framework for Opening Doors and Drawers in Kitchen Environments," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Minnesota, USA, May 2012, p. 3852–3858.

[7] E. Klingbeil, A. Saxena, and A. Y. Ng, "Learning to Open New Doors," in *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, Oct. 2010, p. 2751–2757.

[8] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, ..., and E. Berger, "Autonomous Door Opening and Plugging In with a Personal Robot," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, USA, May 2010, pp. 729–738.

[9] W. Chen, T. Qu, and Y. Zhou;, "Door recognition and deep learning algorithm for visual based robot navigation," in *Proc. IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Legian, Bali, Dec. 2014, pp. 1793–1798.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," May 2016, to be published.

[12] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[13] I. A. Sucan and S. Chitta, "Moveit!" *Online at http://moveit. ros. org*, 2013.

[14] M. Quigley, S. Batra, S. Gould, E. Klingbeil, ..., and A. Y. Ng, "High-Accuracy 3D Sensing for Mobile Manipulation: Improving Object Detection and Door Opening," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009, pp. 2816–2822.

[15] M. Derry and B. Argall, "Automated Doorway Detection for Assistive Shared-Control Wheelchairs," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013, p. 1254–1259.

[16] Y. Zhou, G. Jiang, G. Xu, X. Wu, and L. Krundel., "Kinect Depth Image Based Door Detection for Autonomous Indoor Navigation," in *Proc. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Edinburgh, Scotland, UK, Aug. 2014, p. 147–152.

[17] U. Adar and L. Bayindir, "Door Detection Using Camera Images obtained from Indoor Enviroments ," in *Proc. IEEE Signal Processing and Communications Applications Conference (SIU)*, Vijayawada,India, May 2015, p. 2005–2008.

[18] C. Chen and Y. Tian, "Door Detection via Signage Context-based Hierarchical Compositional Model," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops (CVPR Workshops)*, San Francisco, California, USA, Jun. 2010, pp. 1–6.

Publication P2

Autonomous 3D model generation of unknown objects for dual-manipulator humanoid robots

Adrian Llopart[1,2], Ole Ravn[1], Nils A. Andersen[1], and Jong-Hwan Kim[2], *Fellow, IEEE*

Abstract

This paper proposes a novel approach for the autonomous 3D model generation of unknown objects. A humanoid robot (or any setup with two manipulators) holds the object to model in one hand, views it from different perspectives and registers the depth information using a RGB-D sensor. The occlusions due to limited movement of the manipulator and the gripper itself covering the object are avoided by switching the object from one hand to the other. This allows for additional viewpoints leading to the registration of more depth information of the object. The contributions of this paper are as follows: **1.** A humanoid robot that manipulates objects and obtains depth information **2.** Tracing the hand movements with the robots head to be able to see the object at every moment **3.** Filtering the point clouds to remove parts of the robot from them **4.** Utilizing the Normal Iterative Closest Point algorithm (depth points, surface normals and curvature information) to register point clouds over time. This method will be applied to those pointclouds that include the robots gripper for optimal convergence; the resultant transform is then applied to those point clouds that describe only the segmented object **5.** Changing the object from one hand to another **6.** Merging the resulting object's partial point clouds from both the left and right hands **7.** Generating a mesh of the object based on the triangulation of final points of the object's surface.

No prior knowledge of the objects is necessary. No human intervention nor external help (i.e visual markers, turntables ...) is required either.

Keywords: humanoid robot, 3D model creation, point cloud processing

1 Introduction

With the increase of social and service robotics, the demand for Human-Robot-Object collaboration has risen considerably. Therefore, robots necessarily have to understand

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **(adllo, or, naa)@elektro.dtu.dk**

[2]RIT Lab, Department of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea, **(adllo, johkim)@rit.kaist.ac.kr**

their surroundings to be able to interact with them. Over the past years, special emphasis has been put on robots capable of adjusting to dynamically changing environments, especially when dealing with object recognition and manipulation. Novel research has been proposed for a more rapid and precise detection of known objects. Despite this, robots must also be able to cope with unknown objects: being able to model them becomes a key feature for faster detection, recognition and manipulation in the future.

1.1 Related work

Many approaches to object recognition deal with identifying the 6 DoF pose estimation of the object based on the correspondence grouping of a set of points with a previously generated model [1], [2]. Other approaches use the synthetic data of 3D models to train Convolutional Neural Networks (CNN) for object detection [3], [4], [5]. Finally, some research has been done lately on the generation of grasping poses based solely on the object's point cloud or model [6]. Consequently, a previous 3D model of the model must be known.

Generating 3D models from unknown objects can be accomplished in many ways, each of which has their own advantages and inconveniences. The objects placement, when generating the model, can be divided into three main groups: **a.** static objects with the camera moving around it, **b.** static camera with object on top of a turntable which is rotated, **c.** non-static object and camera.

Concerning the first type, the major issue arises when transforming the camera poses as it revolves around the object. The usage of markers is a widespread solution but requires human intervention, not only for positioning the markers but also to move the camera. The generation of the model can then be achieved by ray-casting the objects silhouette from every view onto a 3D regular grid (volumetric image) as proposed by Denkowski [7]. A more common approach when using markers is to apply an Iterative Closest Point (ICP) algorithm to the point clouds extracted from the depth images of every viewpoint [8].

When dealing with the second type of object placement, namely doing so on a turntable with a fixed camera, the problem with a moving frame disappears. However, the necessity for a human or any other external agent to spin the turntable limits the autonomy of the model generation. Once again, to keep track of the objects viewpoint (in other words, how much the table has rotated), some approaches continue to use markers [9], whilst others rely on SIFT features [10] or the ICP algorithm with loop closure [11].

The last objects placement, whilst being the most difficult to track, allows for the maximum autonomy. Particularly, assuming a robot that wants to autonomously model unknown objects it has recently grasped, to be later used for a faster object recognition and detection. In this case, the objects frame will be inconstant (because the manipulator and gripper holding the object will be moved to try and view it from different perspectives), and so will be the camera frame (presuming the camera is mounted on the head of the robot, it needs to move to be able to track the manipulator's end effector and the object). Krainin *et. al.* [12] propose the usage of a Kalman filter for the camera to keep track of the gripper plus utilizing a modified ICP algorithm (that takes into account sparse feature matching, dense color matching and prior state information) and loop closure to generate smooth object models. Even though this methodology could clearly be considered *state-of-the-art* in terms of autonomously

generating 3D models, it still requires the help of external agents: the problem of grasping an object with a manipulator is that parts of the object will always be occluded. For this reason, the paper suggests leaving the object on a table or any other surface and regrasp it from another position to finish registering the point clouds.

1.2 Basic methodology

The approach proposed in this paper will try to make the registration process of unknown objects as autonomous as possible. The only assumption required is that the robot has an unknown object grasped in its gripper. Plenty of research has been conducted in this field; specifically, a method to achieve this has been previously proposed by Llopart *et. al.* [13].

The pipeline starts with a dual-manipulator humanoid robot holding an object in one hand. The arm and gripper will be moved in a way that **a.** the end effector will always be inside the field of view of the robot whilst the head is also tracking the object (Section 2), **b.** the difference between end poses is very small to allow better NICP convergence (Section 3.2), **c.** the total amount of viewpoints will try to cover the entire 360° around the object.

The RGB-D sensor located in the robot's head will provide depth images for each step, which are then converted to point clouds. The point clouds are then filtered with respect to the distance to the RGBD sensor and to the robot's surface (Section 3.3). A radius sparse outlier process will also take place. The resulting point clouds will then be transformed into the grippers frame to become invariant to movement (Section 3.4). The point cloud's normals and curvatures will be estimated and used during the NICP registration (Section 3.5). One of the major problems previous approaches had was that convergence failed when registering symmetrical or low-detailed objects. To solve this, the NICP is done to the point clouds that include those parts of the robot that are invariant to the object, i.e. the grippers. This allows for additional information and evident better results. The obtained transformations are then applied to the point clouds that represent only the object. The occlusions and lack of viewpoints that occur with one manipulator can be solved by changing the object from one hand to another, and carrying out the same procedure again (Section 3.6). This results in two partial models of the object. These two point clouds must be merged to obtain one complete model (Section 3.7). Finally, a Moving Least Squares (MLS) algorithm is carried out to smoothen the surfaces of the model and remove artifacts. A mesh of it is then built based on a greedy surface triangulation algorithm (Section 3.8).

The experimental setup, results and discussion is found in Section 4. Conclusions and future work are dealt with in Sections 5 and 6, respectively.

2 Tracking the gripper's movements

The key concept proposed in this paper is registering depth information of the object from different perspectives. To achieve this, the robot rotates and translates the object continuously. Thus, it becomes essential that the robot can follow the movements of the end effector instantly with its head, where the depth sensor is located.
Krainin *et. al.* [12] propose the usage of a Kalman filter, taking as input the previous time step mean and covariance, the clouds representing the manipulator and object, and joint angles reported by the encoders of the manipulator.

A different and much simpler approach to achieve gripper tracking is to constantly monitor the position (x, y, z) of both end effectors (left and right grippers). This information must be then transformed into two angles: pan and tilt, corresponding to the 2 DoF the robot's head has. Two simple equations solve this problem:

$$pan\_angle = atan2(y, x) \tag{P2.1}$$

$$tilt\_angle = atan2(z, \sqrt{x^2 + y^2}) \tag{P2.2}$$

This allows the robot to rapidly switch from following the left end-effector to the right one by simply changing the input coordinates to those of the selected gripper.

3 Model creation

This section describes the core concepts of the proposed methodology: the disabling of collision checking between manipulator and object, the multiple filtering steps of the depth data from the RGB-D sensor, the transformation to the grippers frame, the registration of the multi viewpoint data to generate the objects model, changing the object from one hand to another to create two semi-full models of the object, their merging procedure and, finally, the creation of a mesh for the model.

3.1 Disabling collision checking

Before the proposed pipeline can be executed, the collision checking between the objects point cloud and the robots links must be removed. The *MoveIt* libraries for ROS will be used to control the manipulators and to set their poses. An octomap of the environment will be continuously generated to evaluate whether an action can be executed or if the movement will end up colliding with the surroundings. The only issue with this approach is that the object's point cloud (which will be held by the robot) will also be included in the octomap, thus denying any possible robots movement because the gripper will already be in a collision state with the object. Hence, by removing the collision check between the robots links and the part of the octomap that represents the object, the manipulators will be free to move.

3.2 Selecting poses

Due to the robot's configuration and limitations, setting up multiple poses for all viewpoints becomes a tedious task with poor results. For that reason, instead, one manipulator pose is set and two specific revolute joints in the wrist are rotated, as seen in Fig. P2.1. By rotating in a step-wise manner both wrist joints, the robot is able to take 360° depth images of the objects. This allows for better control of the difference between poses, smoother transitions and, most importantly, leaves the object in a quasi-static state, where the center of the object barely moves, thus removing the requirement of a pre-alignment step prior to the NICP procedure. For every rotational step, the filtering and registration procedures take place. Some end effector poses are shown in Fig. P2.2.

Figure P2.1: The red arrows represent the rotational wrist joints.

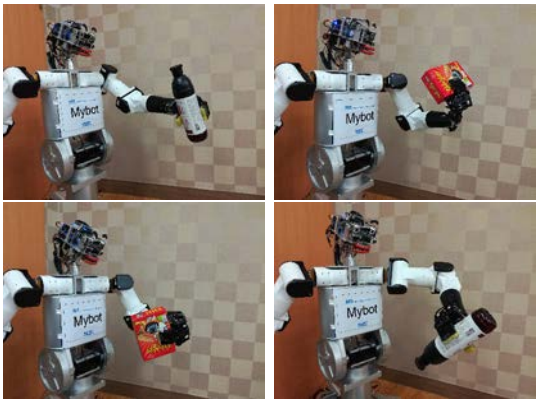


Figure P2.2: Different manipulator poses to see the objects from all possible viewpoints.

3.3 Filtering of sensor data and robot links

Depth images are obtained from a RGB-D sensor and transformed into 3D point clouds. These represent the environment that surrounds the object. Yet, some filtering and segmentation must be done to be able to extract only the important information (the objects shape) from the large quantity of points the sensor outputs. The process is shown inf Figure P2.3.
The first step is to limit the range in which useful information is found: considering that the maximum reach of the manipulators (when they are fully stretched out) is of around 1 meter, it makes no sense to keep point cloud data which is further away because the object is sure to not be outside that region. The depth sensor does not output points closer to 5 cm, thus, if a point is in that range, it is surely an error and must be discarded too. Removing so many points reduces drastically the processing

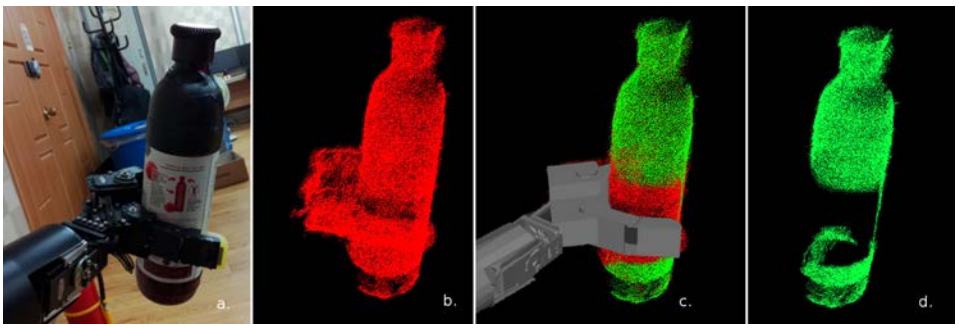


Figure P2.3: From left to right. **a.** RGB image of robot grasping bottle **b.** Full model after registering point clouds from different viewpoints using only one hand. Both object and robot links are included as red points **c.** Simulated gripper on top of point cloud. The green points correspond only to the object's geometry **d.** Final partial model of the object.

time and improves performance in future steps.
Then, those points that represent some parts of the robot will be filtered out too. This means that two point clouds will result from this process. The former will have all points belonging to the robots links removed, thus, only the segmented object data will be visible. The latter, will have most of the robot's link's points also removed, except those that represent the gripper itself. As mentioned in Section 1, modeling symmetrical objects usually ends in bad results during the registration process due to poor convergence when applying the NICP algorithm. For this reason, the entire pipeline proposed in this paper will use point clouds that include the gripper since they add information that removes the symmetry problems when registering clouds.
All small noise and artifacts that are still present in the point clouds are minimized using a radius sparse outlier filter [14]. Those points that do not have a certain minimum number of neighbors inside a radial threshold will be eliminated from the cloud.

3.4 Transforming points to the gripper frame

To be able to register point clouds, it is necessary that they have the same frame and have been slightly pre-aligned before the NICP algorithm is applied. Doing this increases greatly the chances of convergence.
The problem with multi viewpoint registration is that even if the base frame of the data is the same (the camera frame), due to the movement of the end-effector, the point clouds will never be the pre-aligned. To achieve this, an initial alignment based on local feature descriptors (e.g. FPFH) could be applied, but this process might produce bad results in itself, specially when dealing, once again, with symmetrical objects, where key feature density is low.
A simpler solution is to transform all received point clouds to the grippers frame. The advantages of this approach are that the objects points will be invariant to the hands

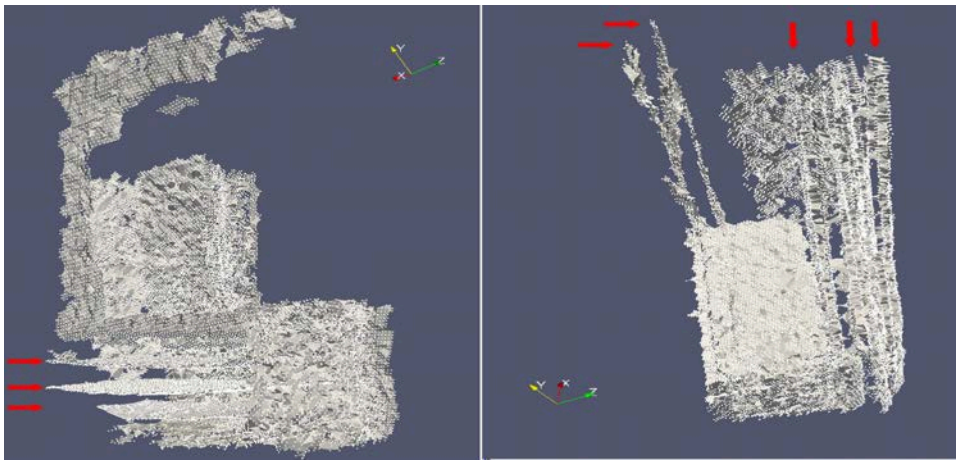


Figure P2.4: Results of building a model without ICP registration: the accumulated drift renders the model unusable. The red arrows show those planes that have translational drift errors. A correct model achieved using registration based on the same data is shown in Fig. P2.5 and P2.6.

movement (because once the object is grasped, it does not move in relation to the hand grasping it) and so no pre-alignment will be necessary. In fact, in the optimal scenario, this solution will allow to build the 3D model without the need for registration. Nonetheless, the reality is that small drift errors will accumulate over time between point clouds rendering the generated model useless (Fig. P2.4). Hence, the registration process becomes a high necessity to be able to correct for those small translational and rotational errors.

3.5 Normal Iterative Closest point (NICP) registration

The backbone of the proposed pipeline is the constant registration of point clouds over time from different viewpoints. These point clouds are the result of the objects segmentation, have the same reference frame (the gripper) and are roughly pre-aligned. To fuse these point clouds in the correct manner, the Normal Iterative Closest Point (NICP) algorithm will be used. NICP minimizes an augmented error metric (based on point coordinates and surface normals and curvature) during the least squares formulation of the alignment problem to find the best data association (transform) between two sets of point clouds. After that, a smoothing process (MLS) is performed to obtain better results.
Generally, the second point cloud will include most of the information from the first, but with additional details due to modifying slightly the perspective. It is important that the changes from one point cloud to the next are very small so that the NICP algorithm can converge. If the differences are too great (for instance, by having rotated the hand a full 180° so that the opposite side of the object is being viewed), the NICP will fail to converge or give poor transforms which will accumulate over time, leading to suboptimal results. This is the main reason for using small rotational steps in the joints when changing the end effector's poses.

For even better results, a surface smoothing algorithm is applied. By estimating the normals of the point cloud's surface, and using the Moving Least Squares (MLS) algorithm as proposed in [11], [12] and [14], the resulting normals will be aligned producing a more precise model of the object, with less noise, occlusions and "double walls" artifacts. The general leaf size during the smoothing process is set to 2 cm, however depending on the geometry of the object, this value must be changed. For objects with sharper sides (e.g. boxes) this values should be lowered (1 cm) so as not to round off the edges too much. For initially already curved surfaces, like bottles or balls, this value can be slightly increased.

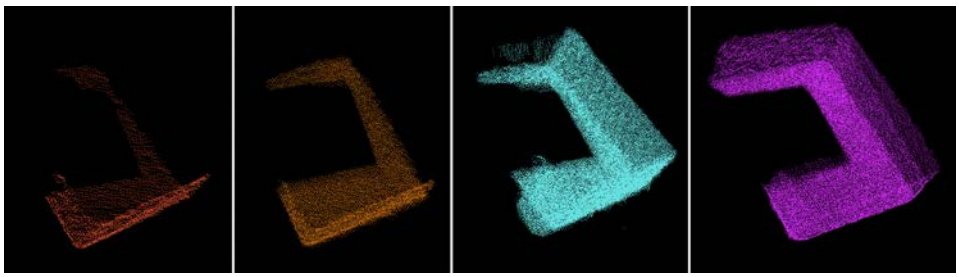


Figure P2.5: NICP registration procedure of a box over several end effector poses.

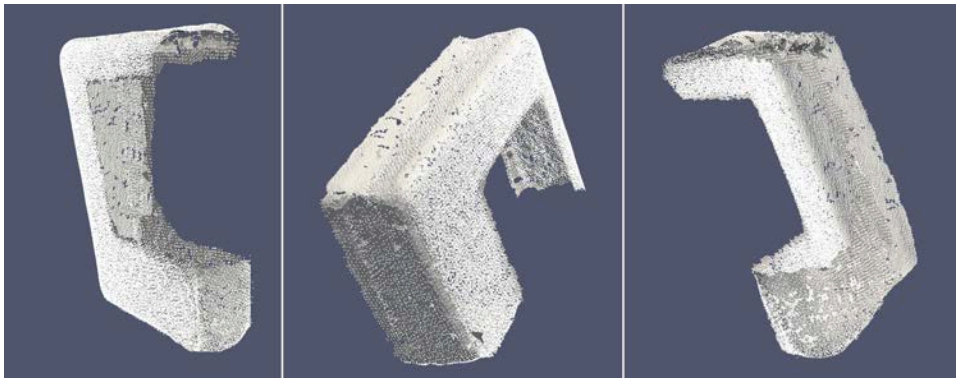


Figure P2.6: Three views of the partial model generated by constructing a mesh from the resulting point cloud in Fig. P2.5

3.6 Changing hands

Occlusions are a big problem when modeling objects. These are due to a limited movement of the end effector (which does not allow the object to be seen all around) or links of the robot always blocking the view of part of the object (specifically the fingers and palm of the gripper).
A way to solve this is by grabbing the object from a different position and starting the registration process all over again. The results from the last registration will be merged with those of the previous registration to achieve a complete model. The fact that the system must not depend on external help, makes the grasp change difficult. If, for instance, a table could be used, the robot would have to simply place the object on it and re-grab it from a different position [12]. In spite of this, a second manipulator can be used: it is important that the second grasp allows for additional viewpoints and for the registration of the object's opposite side. For that, the second grasp must be rotated 180° with respect to the initial one. To do so, and due to the movement limitations of both manipulators, during the exchange one hand will be facing the opposite direction as the other, as seen in Fig. P2.7.b. In the current approach, the poses of the end effectors have been predefined. For an additional degree of autonomy, new grasping poses could be found based on the geometry of the partial model already created using

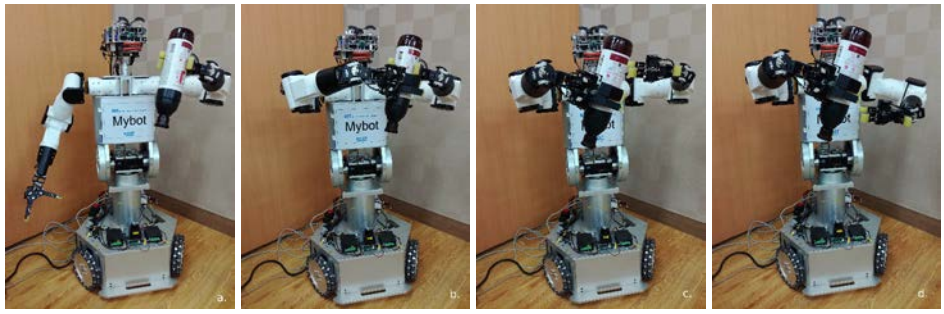


Figure P2.7: *Mybot* humanoid robot changing object (bottle) from left to right hand.

the *agile_grasp* package [6], as seen in Llopart *et. al.* [13].
Finally, for the two sides of the object to be merged together, it is necessary that they are related to the same frame. For this reason, when changing hands, the registered point cloud resultant from the movements of the first gripper must be transformed to the frame of the second gripper.

3.6.1 *Mybot* characteristics:

The proposed pipeline has been tested out on the MyBot humanoid robot, developed in the Robot Intelligence Technology (RIT) Laboratory at KAIST (Fig. P2.2 and P2.7). It includes an Odroid XU board, Ubuntu 16.04 and ROS Kinetic. A Xtion ASUS RGB-D sensor is mounted on the 2 DoF head of the robot. It also has two 7 DoF arms with 3 finger grippers each. Finally, the torso of the robot includes another 2 DoF for panning and tilting.

3.7 Merging both partial models

Having two, almost complete, models of the object (one for each hand used) is not enough. For the model to be useful in future real life scenarios, it is necessary that it represents the object correctly from every angle. For this reason, both point clouds must be merged.
As aforementioned, both point clouds are related to the same frame. Generally, these point clouds will have an offset due to precision errors during the changing hands process. Hence, it is necessary to pre-align them. The Fast Point Feature Histogram (FPFH) descriptors can be calculated for both clouds. In this case, the FPFH based on *OpenMP* is used since its multi-threaded implementation produces results at a faster rate (6-8 times) than the normal FPFH descriptors estimation. When the features from both clouds are matched, a transformation between both models is found that allows them to be roughly aligned.
Then, by using the NICP algorithm, and setting the parameters so that the final convergence is achieved by small rotations and translations, a full and complete model of the object is achieved (Fig. P2.9.c). Once again, the Moving Least Squares algorithm is applied to reduce noise and errors during the merging process.

3.8 Building the mesh of model

For better visualization purposes, a greedy surface triangulation algorithm is run on the resulting point cloud with normals which results in a triangle mesh of the object. The maximum search radius and nearest neighbors values are set to 0.025 and 500, respectively, but may be altered depending on the desired number of resulting triangles. The file will be stored with a *.vtk* format [15]. Some results are seen in Fig. P2.6, P2.8 and P2.9.

4 Experimental results

The experiments were carried out on the *MyBot* humanoid robot. Four different unknown objects (two boxes and two bottles) were placed in one of the grippers and the process was started. The rest of the pipeline was completely autonomous. Some

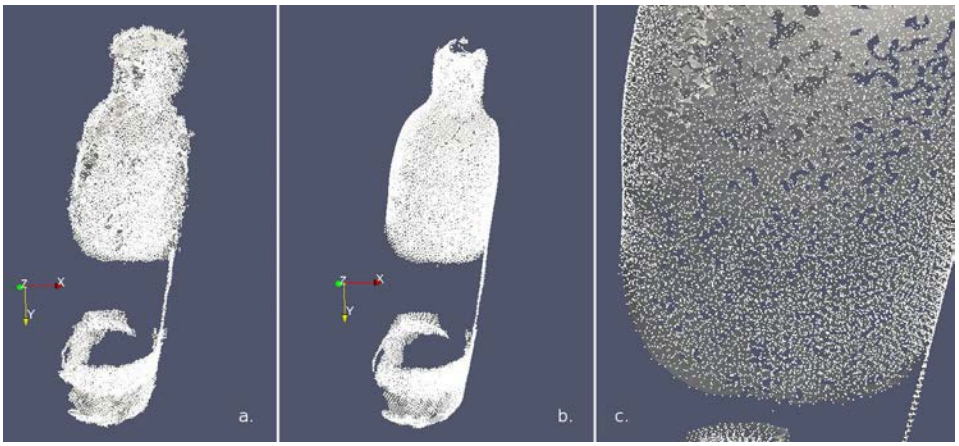


Figure P2.8: Mesh result from Fig. P2.3, from left to right. **a.** Without prior smoothing **b.** With prior smoothing **c.** Detail of mesh.

partial models of objects are seen in Figures P2.8, P2.6, P2.9.a and P2.9.b. Full models are seen in Fig. P2.9.c.

As seen in these figures, the proposed pipeline correctly outputs 3D models of unknown objects. Their overall shapes and sizes are very similar to the real ones, which is a great result considering the geometrical symmetries in all test cases. A way of assessing the overall success of the results is difficult to find. In spite of this, the measurement discrepancies between real object and generated model will be evaluated (Table P2.1). It is seen that the size difference between real objects and models generally stays below the 6 mm mark: likewise, the percentage error never surpasses 9%. Concerning the overall shape of the models, the meshed result clearly matches the real surface of the objects, except for the last model of Fig. P2.9 where the chosen value for the smoothing process was set a bit to high, thus flattening parts of the geometry. Despite obtaining positive results, in some cases occlusions do still occur, hence the lack of details.

A way to achieve better results is by correctly differentiating between points that represent the object or the robot. When filtering the robots surface from the initial point cloud, an overall minimum distance threshold was selected. By fine-tuning the distance between robot and points, more precise results would be achieved. For instance, by reducing the threshold for the finger links, less object points would be filtered out, and, consequently, more details would appear in the final model (the red points in Fig. P2.3.c. would be green).

The major issues when building full 3D models occur when merging the partial results.

Object	Red box			Blue box			Black bottle		Blue bottle	
Dimension	w	h	d	w	h	d	ø	h	ø	h
Object (cm)	12.8	17.5	5.6	8.1	16.9	6.9	7.0	28.7	5.7	18.4
Model (cm)	12.7	17.9	6.1	8.5	17.3	6.6	7.6	28.8	6.1	18.2
Error (cm)	0.1	0.4	0.5	0.4	0.4	0.3	0.6	0.1	0.4	0.2
Error (%)	0.78	2.29	8.93	4.94	2.37	4.35	8.57	0.35	7.02	1.09

Table P2.1: Geometric measurement comparison between real object and model.

Figure P2.9: From left to right. **a.** Partial model created using left gripper **b.** Partial model created using right gripper **c.** Full model after merge **d.** Real RGB image of model (for viewing purposes only).

As aforementioned (Section 3.7), the pre-alignment is done using FPFH descriptors followed by an NICP algorithm. However, if the partial models do not represent almost

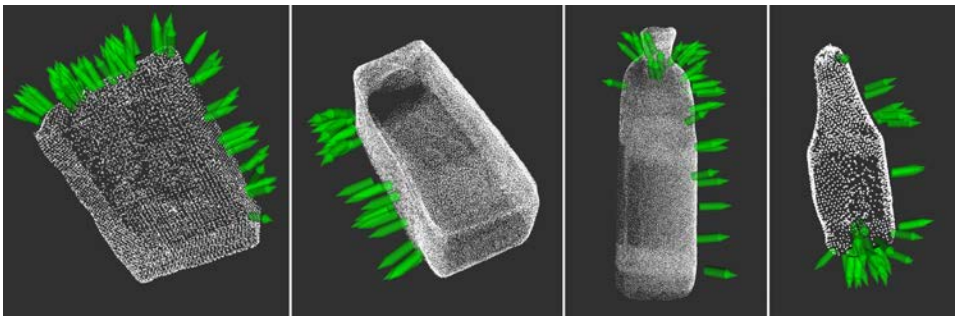


Figure P2.10: Grasping poses for the generated 3D models based off solely the geometry of the point cloud and the grippers dimensions, as proposed by Pas *et. al.* [6]

the entirety of the object, the alignment process will fail, rendering the final model unusable. Additionally, even if the merging process finishes correctly, usually, some parts of the model will still have not been modeled. These normally describe those surfaces of the object that were in contact with the grippers during the registration process, thus, being occluded. Another cause is the mechanical limitations of the manipulators to see the object from certain viewpoints. A good example of occlusion errors (holes in the model) in the final result are the second and fourth models in Fig. P2.9.c.

5 Conclusions

Full 3D models of unknown objects are generated through the proposed pipeline. Registering the point clouds of different views whilst holding the object with one manipulator, enables the robot to create a partial (due to occlusions) model of it. By switching the object from one hand to another and repeating the process, a new partial model is obtained which will not have the occlusion errors of the first one (and vice versa). Merging both partial models accomplishes the removal of errors and the generation of a full 3D model of the object that can be later used for detection, recognition or manipulation purposes. The results show that correct models are being generated for diverse objects in spite of the geometrical symmetries. Fig. P2.10 shows possible grasping locations based off the geometry of the 3D model, which can be stored and applied whenever the object has to be manipulated.

6 Future work

For better results, a loop closure detection system, as proposed in [11], [12] and [16], could be implemented. Additionally, the poses from which the object is seen have been pre-selected by the user. These do not take into account occlusions nor geometry of the object, sometimes leading to redundant point clouds that add no information or, on the contrary, lack of viewpoints to generate a full model. Consequently, it would be interesting to add the *Next best view* concept presented in [12] and [17]. The ultimate goal of the proposed approach is to be combined with the pipeline introduced by Llopart *et. al* [13] to autonomously detect, segment and manipulate unknown objects using a humanoid robot.

Bibliography

[1] A. Aldoma, F. Tombari, L. D. Stefano, and M. Vincze, "A Global Hypotheses Verification Method for 3D Object Recognition," 2012, european Conference on Computer Vision (ECCV) . Lecture Notes in Computer Science, vol 7574. Springer.

[2] M. Zhu, K. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single Image 3D Object Detection and Pose Estimation for Grasping," in *Proc. International Conference on Robotics and Automation(ICRA)*, Hong Kong, China, May 31 - June 5 2014.

[3] K. Sarkar, K. Varanasi1, and D. Stricker, "Trained 3D models for CNN based object recognition," in *Proc. International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 13-16 2015.

[4] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning Deep Object Detectors from 3D Models," in *Proc. International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 13-16 2015.

[5] S. Gupta, P. Arbelaez, R. Girshick, and J. Malik, "Aligning 3D Models to RGB-D Images of Cluttered Scenes," in *Proc. International Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June 7-12 2015.

[6] A. Pas and R. Platt, "Using Geometry to Detect Grasp Poses in 3D Point Clouds," in *Proc. International Symposium on Robotics Research (ISRR)*, Genova, Italy, September 2015.

[7] M. Denkowski, "GPU Accelerated 3D Object Reconstruction," 2013, procedia Computer Science 18, 290–298. Web.

[8] R.-G. Mihalyi, K. Pathak, N. Vaskevicius, T. Fromm, and A. Birk, "Robust 3d object modeling with a low-cost rgbd-sensor and ar-markers for applications with untrained end-users," in *Robotics and Autonomous Systems*, 2015, ch. 66, pp. 1–17.

[9] J. Xie, Y.-F. Hsu, R. Feris, and M.-T. Sun, "Fine registration of 3d point clouds fusing structural and photometric information using an rgb-d camera," in *Journal of Visual Communication and Image Representation*, 2015, ch. 32, pp. 194–204.

[10] T.Foissotte, O. Stasse, A. Escande, P.-B. Wieber, and A. Kheddar, "A Two-Steps Next-Best-View Algorithm for Autonomous 3D Object Modeling by a Humanoid Robot," in *Proc. nternational Conference on Robotics and Automation(ICRA)*, Kobe, Japan, May 12-17 2009.

[11] M. Jaiswal, J. Xie, and M.-T. Sun, "3D Object Modeling with a Kinect Camera," in *Proc. Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Chiang Mai, Thailand, June 6-9 2014.

[12] M. Krainin, P. Henry, and X. Ren, "Manipulator and object tracking for in-hand 3d object modeling," in *International Journal of Robotics Research*, September.

[13] A. Llopart, O. Ravn, N. Andersen, and J.-H. Kim, "Generalized Framework for the Parallel Semantic Segmentation of Multiple Objects and Posterior Manipulation," in *Proc. International Conference on Robotics and Biomimetics (ROBIO)*, Macau, China, December 5-8 2017.

[14] R. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," in *Robotics and Autonomous Systems*, 2008, ch. 56.11, p. 927–941.

[15] C. Marton, R. Radu, and M. Beetz, "On Fast Surface Reconstruction Methods for Large and Noisy Datasets," in *Proc. International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 12-17 2009.

[16] T. Weise, T. Wismer, B. Leibe, and L. V. Gool, "In-hand Scanning with Online Loop Closure," in *Proc. International Conference on Computer Vision Workshops (ICCV Workshops)*, Kyoto, Japan, September 27 - October 4 2009.

[17] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3D object models using next best view manipulation planning," in *Proc. International Conference on Robotics and Automation(ICRA)*, Shanghai, China, May 9-13 2011.

Publication P3

Generalized Framework for the Parallel Semantic Segmentation of Multiple Objects and Posterior Manipulation

Adrian Llopart[1,2], Ole Ravn[1], Nils A. Andersen[1], and Jong-Hwan Kim[2], *Fellow, IEEE*

Abstract

The end-to-end approach presented in this paper deals with the recognition, detection, segmentation and grasping of objects, assuming no prior knowledge of the environment nor objects. The contributions of the paper are as follows: **1)** Usage of a trained Convolutional Neural Net (CNN) that recognizes up to 80 different classes of objects in real time and generates bounding boxes around them. **2)** An algorithm to derive in parallel the pointclouds of said regions of interest (*ROI*). **3)** Eight different segmentation methods to remove background data and noise from the pointclouds and obtain a precise result of the semantically segmented objects. **4)** Registration of the object's pointclouds over time to generate the best possible model. **5)** Utilization of an algorithm to detect an array of grasping positions and orientations based on the geometry of the object's model. **6)** Implementation of the system on the humanoid robot *MyBot*, developed in the RIT Lab at KAIST. **7)** An algorithm to find the bounding box of the object's model in 3D to then create a *collision object* and add it to the octomap. The collision checking between robot's hand and the object is removed to allow grasping using the *MoveIt* libraries. **8)** Selection of the best grasping pose for a certain object, plus execution of the grasping movement. **9)** Retrieval of the object and moving it to a desired final position.

Keywords: object detection, convolutional neural nets, semantic segmentation, pointcloud processing, grasping, humanoid robot

1 Introduction

The usage of robots in human environments has steadily gained popularity in the last years. With rapidly growing research in fields such as service and social robotics,

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **(adllo,or,naa)@elektro.dtu.dk**

[2]RIT Lab, Department of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea, **(adllo,johkim)@rit.kaist.ac.kr**

the demand has risen for robots that can understand the environment around them to be able to interact with it. Namely, it has become essential that a wide variety of objects are recognized and manipulated in real-time.

The proposed approach focuses on the perceptive elements required for a robot to find an object and grab it. Therefore, this paper presents an end-to-end generalized framework for object handling. It starts with the recognition and detection of classes of objects in the RGB domain using a CNN, followed by their segmentation in the spatial domain using point clouds, and ending with the actual planning and manipulation of said objects using a humanoid robot. The framework is a continuation of the work presented in [1]. No prior knowledge of the environment nor the object's model is required.

A Convolutional Neural Network, trained on the COCO dataset, is used to derive bounding boxes around 80 different classes in a live video stream obtained by a robot (3). The resulting *ROI* are then applied to the rectified depth images to generate pointclouds of the detected objects (4.1). However, these results include certain noise and irrelevant background data. Thus, they must be segmented. Eight different methods are applied, depending on the class itself, to be able to extract the best possible representation of the object (4.2). This process is done online and requires no human intervention. At any point in time, a user can select which class the robot has to interact with. The segmented pointclouds of the specified class are registered over time to generate a precise and robust model of the object by removing any possible noise and multi viewpoint errors (5.2). Then, an array of grasping poses is derived based on the geometry of the constructed model. The dimension of the robot's hand are also taken into consideration (5.3). To allow grasping, the object's form must be removed from the continuously updated octomap generated by the robot. To achieve this, a 3 dimensional bounding box is found around the registered object's pointcloud and a *collision object*, from the *MoveIt* libraries, is created. Collision checking between fingers and object is then disabled to allow for the grasping (6.2). The best grasping pose out of the whole array is chosen (which must be achievable) and a trajectory is planned. Since the humanoid robot has two arms, either one can be used to reach the target pose, usually the one that has its hand closer to the object. However if the selected arm fails the grasp, the other one is used. Finally, a humanoid robot moves its arms to reach the goal pose, the object is grasped and placed in a desired position (6.3).

The experimental results are presented in 7, followed by concluding remarks in 8 and possible future improvements in 9.

2 Related Work

One of the presented pipeline's strengths is that it requires no prior knowledge of the environment nor the class models to be segmented. This contrasts with other *state-of-the-art* approaches which utilize geometric verifications to match precomputed models and grasping poses of the objects onto the scene ([2], [3], [4], [5]). Namely, they require offline meshes of all objects; which means that the approach is not generalizable.

Multi-viewpoint of a scene, as opposed to one single view, helps improve robustness of object's models hypotheses ([3], [6], [7] and [8]) and becomes essential when applying SLAM-techniques

Asif *et al.* [9] present a novel method, *STEM*, which encodes the appearance and

structural characteristics of an RGB-D point cloud in terms of five feature maps. It allows for RGB-D object recognition and grasp detection using hierarchical cascaded forests, without predefined object models.

3 CNN for object recognition and detection

The pipeline presented in this paper initiates with the usage of a Convolution Neural Network to recognize and detect specific pre-trained classes from images. The goal is to extract *ROI's*, in real time, from a live video stream coming from the robot. Hence, limiting the data into useful class information.

The proposed CNN must be able to predict bounding boxes around classes based on their probability extremely quick and with a high degree of reliability and precision. This optimization is achieved by employing a unified architecture: the YOLO Real-Time Object Detection System ([10], [11]).

It is important that model is accurate, real-time and able to detect multiple objects simultaneously. To attain these goals, the *yolo-coco* model is used. It has been trained on the COCO dataset provided by Microsoft, which includes 80 different classes. However, in this paper special emphasis is put on kitchenware, food and office supplies.

A great benefit of this system is that it is easily integrated with *ROS*. On a GeForce GTX 1070 it can run at an average of 16 FPS. The result given by the detection system includes not only the bounding boxes around the diverse classes, but also the probabilities for each one. Some results can be seen in Fig. P3.1, P3.2, P3.7 and P3.8.

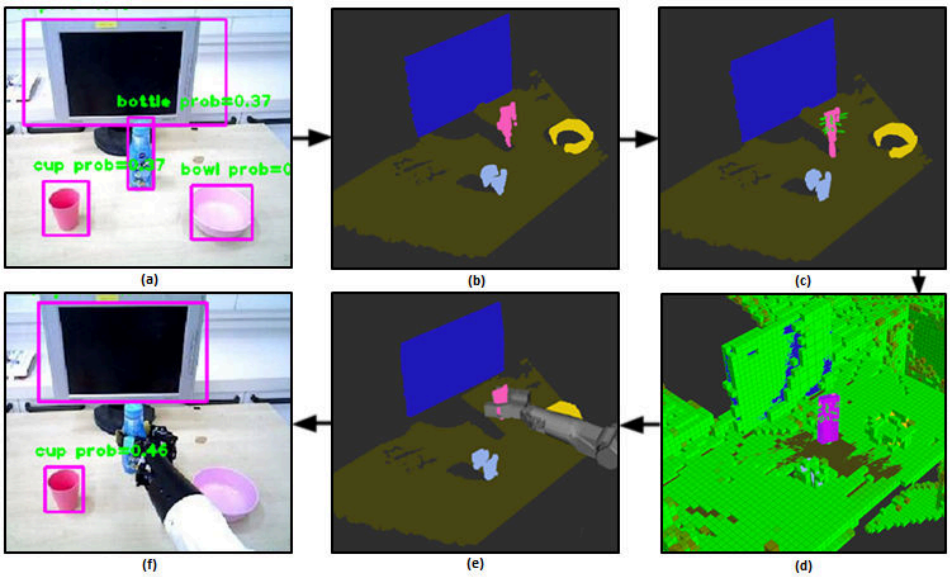


Figure P3.1: End-to-end pipeline of the presented approach: **(a)** Object detection using CNN **(b)** Segmented objects **(c)** Array of grasp poses **(d)** Collision object and octomap **(e)** Planning to target pose **(f)** Execution of movement

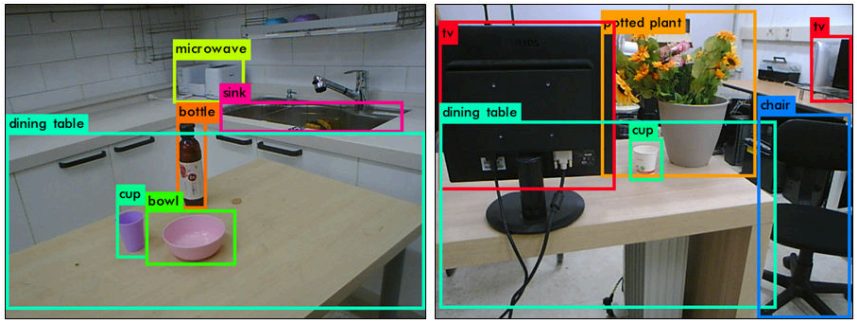


Figure P3.2: Results after CNN is applied to images collected by *Mybot*.

4 Semantic segmentation of objects

Once the classes have been recognized and detected in the colour domain, the knowledge learnt (i.e. the semantic labelling of objects) must be transferred into the spatial domain, that is, segmenting 3 dimensional representations of the objects.

4.1 Generation of pointclouds based on Region of Interest

Initially, the meta data (location and size) of the *ROI's* found by the CNN is transmitted to the depth domain. This can be done because the RGB and Depth images have been previously rectified, using *ROS Indigo* libraries, and match pixel-to-pixel. By removing all data from a depth image which is not inside the *ROI*, performance of the system and quality of the results are substantially increased. Pointclouds of each object are created based on the cropped depth images; one pointcloud for every object detected. These pointclouds will encode not only the necessary data in a pointcloud message, but also the label of the object it represents. The new message type will be called *LabelledPointcloud*.

4.2 Multiple segmentation methods

Even though the objects have been generated in the spatial domain, they still render useless for a correct interaction with the robot. The reason is that these pointclouds generally include noise and background data. Therefore, to obtain the best estimate of the object, these pointclouds must be segmented.

A total of 8 different segmentation methods are employed in this paper. They have been applied using *PCL* libraries. The algorithms are:

1. Planar segmentation
2. Planar segmentation (only vertical surfaces)
3. Planar segmentation (only horizontal surfaces)
4. Outlier segmentation from planes
5. Cylindrical segmentation
6. Spherical segmentation
7. Region growth segmentation
8. No segmentation

The name of the methods does indeed describe very clearly what they do. For instance, the planar segmentation finds all the points within a point cloud that support a plane model. The second and third methods restrict the found planes to only vertical and horizontal surfaces, respectively. Additionally, the vertical planar segmentation method includes also a handle segmentation because some objects that fall into this category require it (for instance refrigerators), similarly to [1]. To achieve this, only points that fulfill the conditions of not being inliers in the plane, being located between the plane and the robot (i.e. not behind the plane) and being less than 10 cm in front of the plane, will be segmented as handles.

The fourth method, goes one step further as the first one. It is used to segment objects that are found always on top of desks or other surfaces (also known as *tabletop* objects). The method finds planes but only stores the remaining points (*outliers*). This method has been enhanced to save only points located between robot and plane, eliminating everything else.

In the cylindrical and spherical segmentation methods, cylinder and sphere models are fitted respectively into the data. In the region growth method, the points that are close enough in terms of the smoothness constraint are merged forming multiple clusters. The largest cluster will be selected as the best descriptor of the object.

The decision of which algorithm has to be applied to which object is done *a priori* following logical criterion. Special emphasis is put on the fact that this is the only part of the whole pipeline that has been hard coded by the authors. Objects such as bottles and cups will obtain better results using a cylindrical segmentation. Monitors, laptops and refrigerators should use the vertical planar segmentation method. Sports balls, oranges and apples work best with spherical segmentation. Objects with no predefined form, like animals or backpacks will be segmented using region growth. And this is done for each of the 80 classes the CNN is able to recognize.

Hence, given an incoming *LabelledPointcloud* message, the label is checked and the correct segmentation method is selected. The result is a second *LabelledPointcloud* message which differs from the first in that all noise and unnecessary background data has been removed. Results can bee seen in Fig. P3.1(b, c, d), P3.4, P3.6(b) and P3.8(b).

It is worth mentioning that the segmentation of pointclouds is done in parallel thanks to the *ROS* interface. Individual nodes have been created for every distinct segmentation method to allow parallel processing, therefore the results update at a much faster rate, allowing real time processing.

5 Derivation of Grasping poses

The interaction between robot and objects is one of the key features presented in this paper. As aforementioned, several objects can now be detected and segmented correctly from the environment, which opens the possibility of them being manipulated by a robot. To achieve this, an array of correct and precise grasping poses (positions and orientations) must be obtained so the robot can grab the object and move it around. These poses will be found depending on: (1) the geometry of a model of the object, which is created by registering segmented pointclouds of the object over time, and (2) the form of the robot's hand.

5.1 User input for the selection of object

The first step to interacting with an object is, indeed, selecting it. In the presented approach, this decision is made by the user, with a simple command stating the name of a class. This can be done at any point in time since the recognition and segmentation of classes is done continuously. However, in future iterations of the proposed pipeline, this decision might be made by the robot itself using artificial intelligence methods.

5.2 Registration of pointclouds to create model of the object

After the user chooses a specific class to handle, its grasping poses must be found. These will be highly dependant on the geometry of the object's model, which must be extremely precise. Since there is no prior knowledge of the object's form, it is essential to create the model online based on the segmented pointclouds.

The major issue with the segmented pointclouds is that they vary greatly depending on the viewpoint of the robot; additionally, they sometimes include noise. Thus, the results may vary from one iteration to another. This problem can easily be turned into an advantage by registering the pointcloud over a certain amount of time. This will allow for the removal of noise and the generation of a model based on pointclouds from multiple viewpoints. In other words, the more the robot looks at and object, and from different positions, the better and more definite model of it is achieved.

The registration will be accomplished using the Iterative Closest Point (ICP) algorithm. Pointclouds will only be registered if the transformation between one and another is smaller than a certain threshold. Due to this, noise is easily removed and, most importantly, pointclouds that have been wrongly segmented will be discarded. If the difference between the current registered pointcloud and following new points is too big, the registering process is halted and restarted.

All in all, the resulting pointcloud will increase over time as more and more correct points are registered, creating a robust and precise model of the object. The registration process can be visualized in Fig. P3.3. Additional results for the registered pointcloud can be seen as red pointclouds in Fig. P3.4 and inside the generated *collision object* in Fig. P3.6(d).

Figure P3.3: Registration over time of different pointclouds (grey) representing a cup. The result appears in red. Its growth is clearly visible over time as more and more points become registered.

5.3 Array of grasping poses based on geometry of the object

Much emphasis has been put on creating the best possible model of the object because the grasping poses will be highly dependant on its geometry. This is due to the fact that the *agile_grasp* package [12] is used. Moreover, the algorithm also takes into account the dimensions of the robotic hand.

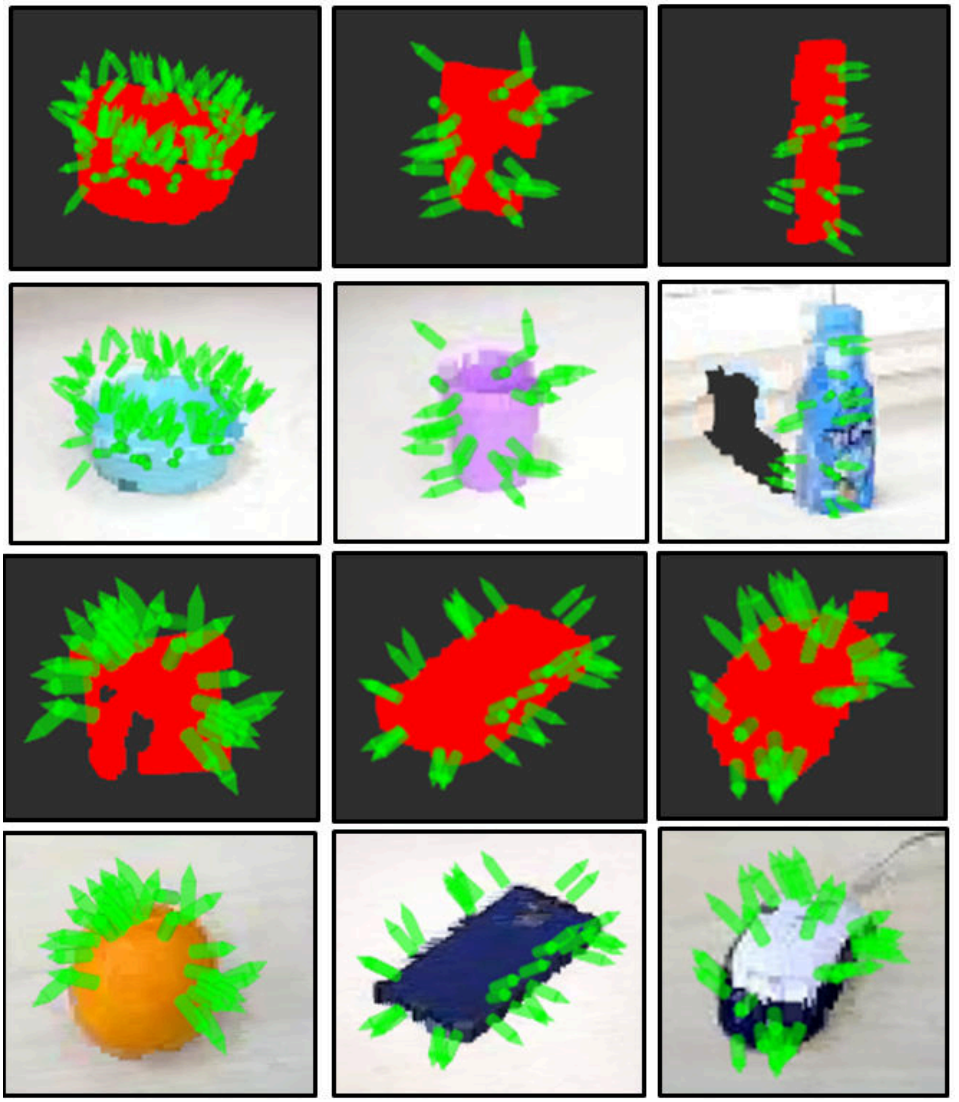


Figure P3.4: Array of grasp poses (green vectors) derived from the registered pointclouds of objects (red). These results are also shown overlapped on top of the original coloured pointclouds: cup, bowl, bottle, orange, cell phone and mouse, respectively

The usage of this approach is fundamental for the presented framework. One of the key concepts shared by both projects is that there must be no prior knowledge of the manipulated objects, as opposed to other *state-of-the-art* methods that include offline models of the objects and predefined grasping positions (i.e. [3] and [6]). The grasping algorithm [12] has two steps. The former, sampling a large set of grasping hypothesis based on the condition that for a grasp to exist, the hand must be collision-free and part of the object's surface must be contained between the fingers. The latter, using machine learning (Support Vector Machine) to classify the validity of those hypothesis checking whether the grasping points are *antipodal*. Thus, Pas *et al.* [12] report an average grasp success rate in their experiments of 88% when grasping novels objects presented in isolation and 73% when presented in dense clutter. Even though the methodology has been developed for 1 DOF , two finger hands, the results are reliable enough to still use this method with a 3 DOF, 3 fingered hand with which *Mybot* is equipped (6.1).

Some results of good grasping positions for diverse objects can be seen mainly in Fig. P3.4, but also in Fig. P3.1(c).

6 Implementation on humanoid robot

The proposed framework has been carried out on the humanoid robot *MyBot*. The pipeline has been expanded (as shown in Sections 6.2 and 6.3) to be able to port the entire system onto the robot, namely recognition, detection, segmentation and manipulation of objects.

6.1 Description of the robot

MyBot is a robot developed in the Robot Intelligence Technology (RIT) Laboratory at KAIST. It makes use of an Odroid XU board, Ubuntu 14.04 and *ROS Indigo*. A *Xtion ASUS* sensor is employed to obtain rectified RGB and Depth images of the surroundings. It is located on the head of the robot, which includes 2 *DOF* to look around. The robot has a mobile platform for navigation and two 7 *DOF* arms with 3 finger hands. Additionally, the torso of the robot includes another 2 *DOF* for panning and tilting.

Figure P3.5: *MyBot* humanoid robot grasping a bottle.

6.2 Removal of collision checking between fingers and object

An octomap is a 3D occupancy grid mapping approach that models environments into occupied, free or unknown areas. This becomes essential for robot navigation, that is, knowing where certain actions can take place and where the robot will surely collide with obstacles.

A problem of this approach is that the segmented objects will have surely been encoded as occupied space. This means that the system might fail at planing movements to the grasping positions since collisions may appear between fingers of the robot and objects whilst moving. Additionally, the system will certainly fail when grasping the object because grasping obviously requires contact.

The way to solve this is to remove the collision checking between object and fingers. This can be done only because the proposed approach [12] minimizes contacts during movement through an optimal grasping pose array and the *MoveIt* libraries. To remove collision checking, a *collision object* must be created first around the object. It could be created from an object's mesh, but there is no prior knowledge of it. Thus the only solution is to create it from primitive shapes, specifically rectangular prisms.

The proposed contribution is as follows: since the geometry of the object is mostly known due to the registered model of the object, a 3D bounding box can be generated around it. Firstly, the centroid, the normalized covariance and the eigenvectors of the registered pointcloud are found; these represent the principal directions. Then all points are moved onto the found reference frame. The maximum and minimum point values for every axis can be easily found: these represent the width, height and depth of the bounding box. All results are finally converted back to the original frame.

The values obtained from the bounding box are used to create a rectangular prism *collision object*, which removes all points inside it from the octomap. This can be seen in Fig. P3.1(d) and P3.6(d). Lastly, the collision checking between fingers and the newly created *collision object* is removed, which allows the robot to plan a trajectory to the object and finally grab it.

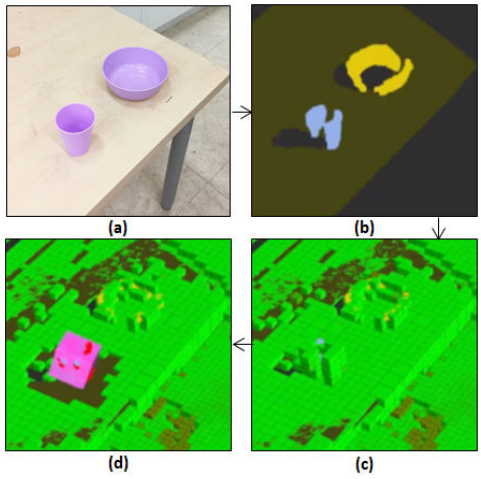


Figure P3.6: Creation of a collision object based on the 3D bounding box around the registered pointcloud (red): **(a)** Original RGB image **(b)** Segmented objects **(c)** Current state of the Octomap **(d)** Generated collision object (pink) removes points from octomap

6.3 Manipulation of objects using *MoveIt* libraries

Interaction between segmented objects and the robot is the final step of the framework and, most importantly, requires all previous sections to have worked correctly. At the end of the day, what truly matters is that when the user states an object's name, the robot is able to, autonomously, find it in a cluttered new environment and be able to pick it up.

Since the robot has two arms, it is important to select carefully which one to manipulate the objects with before even starting to plan a trajectory. Naturally, the hand that is already closer to the object should be the one used because it will require less planning time. For this reason, the positions of all the found poses in the grasp array are averaged and the euclidean distance between that result and the reference frame of both hands is calculated. The hand with the shortest distance will be used to grab the object.

In spite of this, sometimes, selecting the closest hand to an object might not result in the optimal solution. Especially if there are obstacles to avoid between the end and goal poses. This is why, if the planning with the first arm fails, the second arm is utilized. Generally, one of the plans will succeed; however, if both of them do fail, a new perception iteration takes place. Namely, registration of pointclouds, grasp derivation, generation of collision object and planning. This occurs until the object has finally been grabbed or the user halts the process. Further robot cognition could be added at this point to allow more autonomy and better results, like the one presented by *Veiga et al.* [13]. Another example of this is changing the robot's position with respect to the object if the trajectory planning has failed several times and then approach it from a completely different perspective.

Assuming the planning to the grasping pose has succeeded, the robot will execute the movement using the *MoveIt* libraries. This leads to the robot's gripper being positioned right next to the object. The last step is to simply close the gripper and move the object to a desired position or perform a specified task. The created *collision object* is attached to the robot so that when moving the object around, it never crashes into obstacles. The end-to-end perceptive and manipulative framework has, therefore, finished. Various grasping poses, achieved using one or the other arm, for different objects can be seen in Fig. P3.1(f), Fig. P3.5 and, specially, in Fig. P3.7.

7 Experimental results

The proposed framework has been tested in two real life indoor scenarios: a kitchen and an office. The conducted experiments are divided into two blocks: the former, the object perception block, which includes detection and registration of objects via CNN and the generation and segmentation of the corresponding pointclouds. The latter, the grasping block, includes the pointcloud registration of a specified object, the derivation of an array of grasping positions, the creation of a *collision object* based on the 3D bounding box around the registered pointcloud, the selection of which arm to use and, finally, the execution of the grasping trajectory with a humanoid robot.

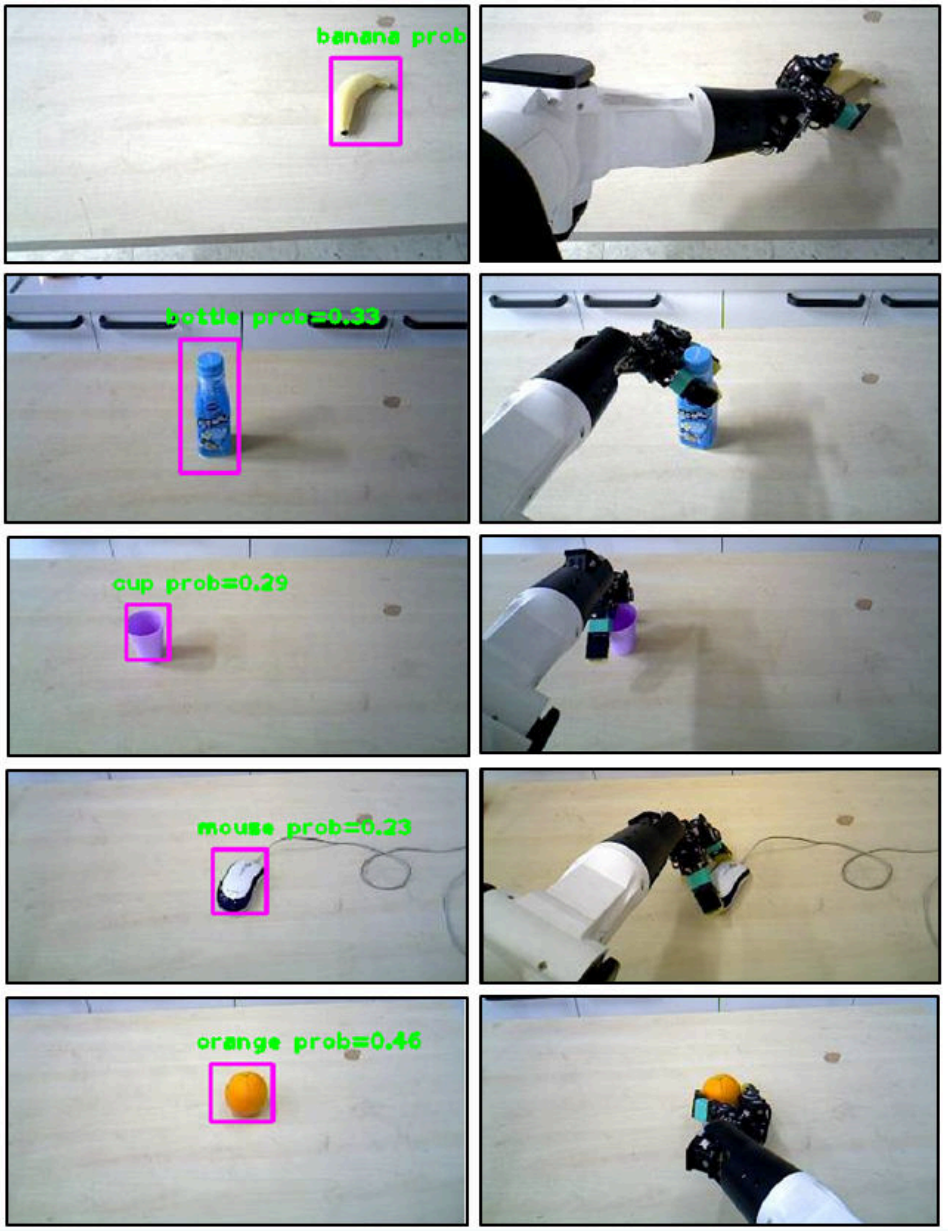


Figure P3.7: *Left column:* Results from CNN applied to several classes. *Right column:* Grasping poses achieved by the humanoid robot.

	Class	Trials	Recognition (%)	Grasp (%)
Single	Banana	10	100	20
	Bottle	10	100	90
	Bowl	10	80	20
	Cup	10	90	80
	Mouse	10	70	50
	Orange	10	90	80
Clutter	Bottle	10	100	80
	Cup	10	100	70
	Orange	10	80	70
	Chair	10	90	-
	Monitor	10	100	-
	Refrigerator	10	90	-
Average			**90.1**	**62.2**

Table P3.1: Results for the recognition and grasping blocks for 9 different classes, both in single-object and cluttered scenes.

7.1 Experimental Setup

A correct object recognition occurs when the object's bounding box is precisely found in the RGB domain, with a proper semantic labelling. This must be followed by a strict segmentation of at least half of the object. Similarly, a good grasping of the desired object occurs when the object's geometry is correctly found using pointcloud registration, a legitimate *collision object* is generated, several grasping poses are derived and the robot plans a trajectory to one of them to ultimately grab the object. All steps in each block must be fulfilled for the test to be considered positively.

The conducted tests are divided into 2 types: single object scenes and cluttered scenes. The reason is to test how highly dense scenes affect both the perceptions of the objects and the manipulation. The classes used in this experiment are also divided into two groups: those that can be both recognised and grasped (cup, bottle, bowl, banana, orange, mouse), and those that can only be recognised (chair, monitor, refrigerator).

For single object perception, the relative position between robot and objects will be changed randomly after every trial. This will help evaluate the system in front of different viewpoints and trajectory distance until grasp. Equivalently, in cluttered scenes, multiple objects will be arbitrarily relocated in the scene. However, special care will be taken so that the goal object is still visible, at least half of it, so that occlusions do not affect the results.

7.2 Results, causes and consequences

The most important conclusion to be extracted from the experiments is that the recognition block achieves highly positive results. With a **90.1%** of correctly recognized, detected and segmented objects from the COCO Dataset, the approach presented in this paper is proven to be competitive in front of other *state-of-the-art* methods. Comparatively, Asif *et al.*[9], using their STEM methodology, achieves results for object recognition of 93.0% success rate.

Concerning the grasping block, the results (**62.2%** grasp success rate) show that some objects can be handled quite easily, whilst others tend to cause more failures. There are 3 main reasons for this: goal position and orientation errors, slippery surfaces and collisions prior to the grasp.

Firstly, if the final grasping pose is not precise enough due to noise in the registration process, then even when the robot reaches the goal position, it will end up either grasping thin air or grabbing inconsistently a small part of the object. Similarly, if the orientation for grasping is not reliable enough, the fingers of the robot might end up slightly touching the object when moving to the goal position and pushing it out of the way. A clear example of this is when trying to grasp cups and bowls from the top. For it to work, the thumb has to end up on one side of the vertical surface of the object, whilst the rest of fingers have to be placed on the opposite side. if the orientation is not precise enough, the fingers will collide with the top of the object and the grasp will be unsuccessful.

The second problem leading to grasping failures is the lack of friction between fingers of the robot and certain objects. If the surface is smooth, the robot will commonly grab the objects but it will just slip away when lifting it. This becomes evident with objects with polished surfaces such as mouses or bowls. In addition, these objects have convex surfaces which makes the whole process of grabbing them even more difficult.

The last issue when grasping objects is small collisions occur between the fingers and the environment before grasping. This problem becomes evident when dealing

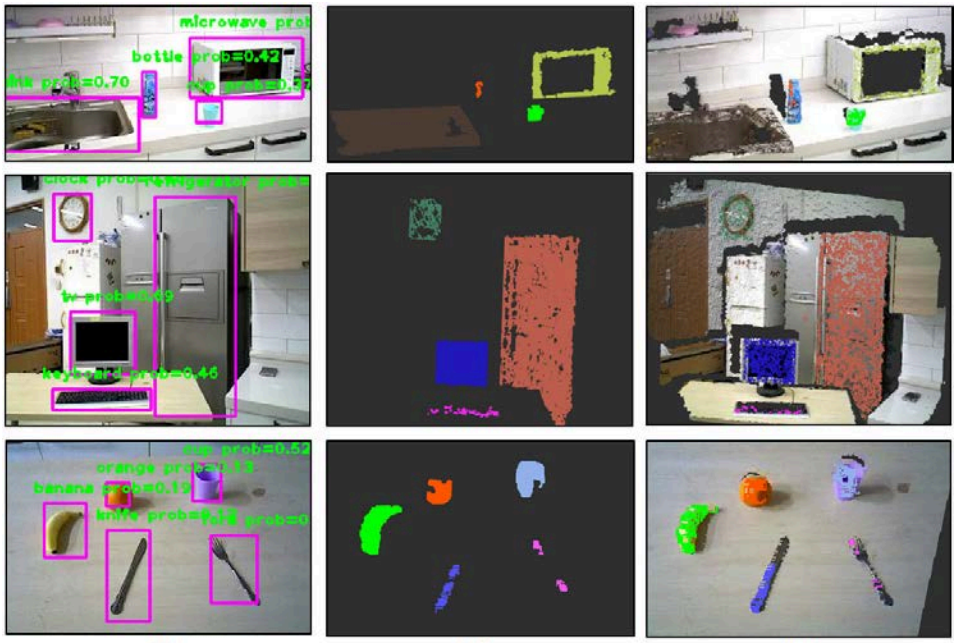


Figure P3.8: Experimental results: **(a)** Object detection using CNN **(b)**, Segmented objects **(c)**, Overlap over original pointcloud

with objects that have small grasping surfaces. For instance, bananas and mouses can be handled only in parts of their surface that are really close to the table they usually are located on. The experiments show that even when the robot correctly positions the hand around the object, when trying to grab it, the fingers collide with the table, blocking the grasp.

8 Conclusion

The presented framework includes the whole pipeline for object manipulation with a humanoid robot: from the detection of objects in the RGB domain, to their segmentation and, finally, their manipulation.

The approach differs from many *state-of-the-art* due to the fact that no prior knowledge of the environment or objects is required. This means that no pre-computed models and grasping poses are known. This is precisely why the proposed methodology can be applied to multiple objects (which, however must fall into one of the 80 classes recognizable by the CNN) independently of their size, form or colour. The system can be greatly expanded by applying, at the beginning of the pipeline, a larger CNN capable of detecting even more classes. The usage of several segmentation methods and the registration over time of the object's pointcloud, allows building a consistent model of the unknown object. As aforementioned, the grasping poses are determined based on the online generated geometry of the object. The downside of this is that the grasps will be of a poorer quality than if they were pre-selected. In spite of this, the approach has the important upside that it is generalizable in front of any object's geometry, making it very powerful when dealing with unknown objects.

The experimental results have proved that the system works consistently from end-to-end. The recognition block achieves highly competitive results whilst the grasping block obtains good results, specially considering the lack of prior knowledge of the objects, Despite this, the grasp pose estimation could be improved using alternative methods. It is worth mentioning that trying to grasp objects with a simpler shape and larger surface, such as boxes, than the proposed objects would entail an increase in the successful grasping trials. This is due to finding more robust grasping poses and the non-existance of slippery surfaces and collision with the environment. This is seen in the fact that cups and bottles (cylinders) and oranges (sphere) have higher success rates than the rest of objects.

All in all, the final outcome is a system that allows a humanoid robot to find certain classes in an unexplored environment, segment the objects from the scene, calculate correct poses and, finally, manipulate them.

9 Future improvements

Even though the presented framework achieves good results considering its performance in front of novel objects and surroundings, there is always room for improvement. Right now the system can detect up to 80 classes which is a good number but not large enough for a real life implementation. *Yolo9000* [11] is a CNN model trained to recognize and detect up to 9000 different classes with real time performance. It would be a very nice addition to the presented pipeline. It would surely mean including many more segmentation methods (like the one presented in [14]) or combinations

of them to be able to obtain adequate results for the large number of new classes (e.g. colour-based region growth segmentation).

At the moment, the selection of the object to grasp is done via user input. This could change with some sort of robot cognition which selects objects based on the future task to be executed, as proposed by [13]. Additionally, when registering the selected object's pointcloud, much better results would be achieved with a multi-viewpoint perspective since a more reliable model would be generated and so more consistent grasping poses would be found (as presented in [3], [6], [7] and [8]). Similarly, if planning to a grasping pose has failed several times, approaching the object from a new perspective might solve the problem. These methods require some level of artificial intelligence.

A force-torque sensor could be added to the fingers to be able to know precisely if the grasp has failed or, otherwise, how good the quality of the grasp is. Finally, SLAM techniques and loop closure could be added to generate full semantic maps over time, as presented in [2] and [6].

Bibliography

[1] A. Llopart, O. Ravn, and N. Andersen, "Door and Cabinet Recognition Using Convolutional Neural Nets and Real-time Method Fusion for Handle Detection and Grasping," in *IEEE International Conference on Control, Automation and Robotics (ICCAR)*, Nagoya, Japan, April 2017.

[2] K. Tateno, N. Navab, and F. Tombari, "When 2.5d is not enough: Simultaneous reconstruction, segmentation and recognition on dense slam," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 16-21 2016.

[3] C. Li, H. Xiao, K. Tateno, F. Tombari, N. Navab, and G. Hager, "Incremental Scene Understanding on Dense SLAM," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, October 9-14 2016.

[4] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. D. Stefano, and M. Vincze, "Multimodal Cue Integration through Hypotheses Verification for RGB-D Object Recognition and 6DOF Pose Estimation," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6-10 2013.

[5] S. Gupta, P. Arbelaez, R. Girshick, and J. Malik, "Aligning 3D Models to RGB-D Images of Cluttered Scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, USA, June 7-12 2015.

[6] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," [ArXiv]. Arxiv, 7 pp., 7 pp.

[7] K. Tateno, F. Tombari, and N. Navab, "Real-time and scalable incremental segmentation on dense slam," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, Sept 28 - Oct 14 2015.

[8] A. Djelouah, J. Franco, and E. Boyer, "Multi-View Object Segmentation in Space and Time," in *IEEE Conference on Computer Vision and Pattern Recognition (ICCV)*, Sydney, Australia, December 3-6 2013.

[9] U. Asif, M. Bennamoun, and F. Sohel, "RGB-D Object Recognition and Grasp Detection Using Hierarchical Cascaded Forests," *IEEE Transactions on Robotics*, vol. PP, no. 99, pp. 1–18, January 2017.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Boston, USA, p. 779–788, June 2016.

[11] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," Dec. 2016, arXiv:1612.08242.

[12] A. Pas and R. Platt, "Using Geometry to Detect Grasp Poses in 3D Point Clouds," in *Proc. International Symposium on Robotics Research (ISRR)*, Genova, Italy, September 2015.

[13] T. Veiga, P. Miraldo, R. Ventura, and P. Lima, "Efficient Object Search for Mobile Robots in Dynamic Environments: Semantic Map as an Input for the Decision Maker," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, October 9-14 2016.

[14] U. Asif, M. Bennamoun, and F. Sohel, "Unsupervised segmentation of unknown objects in complex environments," in *Autonomous Robots*. SPRINGER, 2015, ch. 40 (5), p. 805–829.

Publication P4

Online Semantic Segmentation and Manipulation of Objects in Task Intelligence for Service Robots

Adrian Llopart[1,2], Ole Ravn[1], Nils A. Andersen[1], and Jong-Hwan Kim[2], *Fellow, IEEE*

Abstract

Task Intelligence is the capacity of a robot to learn, reason and execute specific behaviours based on its environment. In this paper, the Task Intelligence problem formulated by the Robot Intelligence and Technology Laboratory at KAIST is researched further: specifically the proposed contribution is a brand new perceptual pipeline in which the recognition, detection, segmentation and grasping of objects is achieved assuming no prior knowledge of the environments arrangement nor the objects appearance. A Convolutional Neural Net (CNN) is used to detect, recognize and semantically label those objects that need to be interacted with. 3D point clouds, corresponding to the objects model, are extracted after several segmentation procedures and registered over time. Dimensional and positional information of the object is acquired. Additional grasping pose data is calculated. All of the collected knowledge is parsed so that the Task Intelligence system is able to deal with previously unknown objects in dynamic environments. This system is formed by an Episodic Memory (Deep-ART), an action sequence generator (FF-planner) and a trajectory warping module for pre-learnt behaviours. The proposed approach has been tested using the *Webots* simulator.

Keywords: Task Intelligence, semantic segmentation, CNN, Episodic Memory, Deep-ART, humanoid robot

1 Introduction

The major difference between industrial and service/domestic robotics is the ability of the latter to correctly determine which tasks are necessary to complete a certain goal and in what sequence, depending on the situational context. To accomplish this, robots

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **(adllo, or, naa)@elektro.dtu.dk**

[2]RIT Lab, Department of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea, **(adllo, johkim)@rit.kaist.ac.kr**

must be able to memorize task sequences and retrieve them when required. However, when a task sequence is needed, but has not been previously learned, a deterministic search method is employed to infer the best actions or behaviours to achieve its goal, thus creating a new sequence altogether. The results are then stored in memory for later use. Finally, these actions must also be adaptive, meaning that they must be able to be altered depending on the object's position and robot's state.

Previous papers that deal with Task Intelligence rely on the robot having all the information of the object's location and orientation since the very start ([1], [2], [3]); otherwise, a CNN [4] is employed to find the objects positions. Nevertheless, the general dimensions of the object and the grasping poses are still manually predefined. This CNN is trained over the images of the specific objects dealt with in the experiments so it cannot be generalized to other objects with different forms, colours and sizes.

This paper presents a novel perceptual pipeline that allows a robot to update online the known, or lack of, data of the objects found in its environment. The information is stored in memory and is used by the modules in the Task Intelligence system to achieve a certain goal. This way, the robot can deal with unknown or highly dynamic environments instead of being confined to a predetermined static scenario.

The paper is organized as follows: Section 2 describes the different elements prerequired for the Task Intelligence system. The proposed perception pipeline is shown in Section 3. Section 4 details the experimental setup and results. The concluding remarks and future work are found in Section 5 and 6, respectively.

2 Task Intelligence systems

In this section, a summary of the modules that form the Task Intelligence system is described. These are the Episodic Memory (Deep-ART), the task generator (FF-planner) and a trajectory warping algorithm.

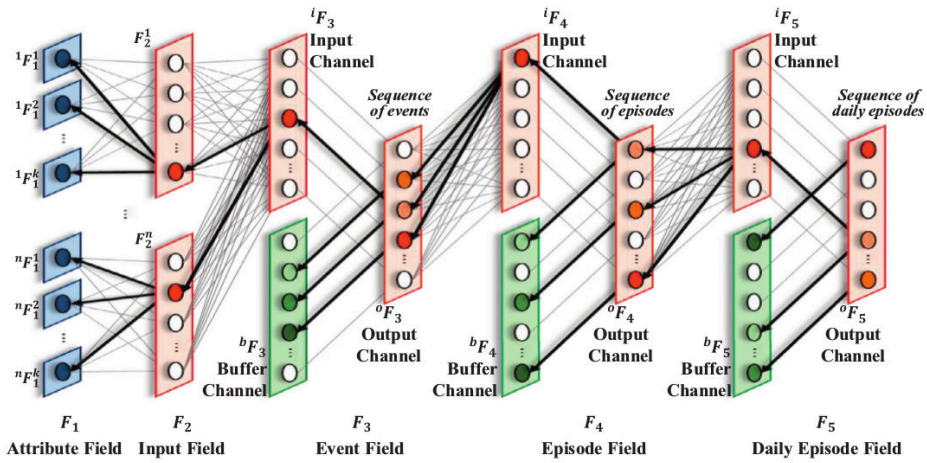


Figure P4.1: Deep-ART architecture

2.1 Episodic Memory: Deep-ART

The Episodic Memory models the long-term memory of humans: a robot can memorize task sequences based on experiences which are later retrieved depending on partial input cues. The unsupervised neural network Deep-ART, proposed by *Park et. al.* [1] will be used (Fig. P4.1). It is based on Fusion-ART [5] and is able to learn and retrieve episodes without errors, as opposed to other approaches ([6], [7]).
The episodes are encoded by categorizing events or task sequences defined as a verb clause {v (action), o_1 (object), γ (geometrical relation), o_2 (object)}.

2.1.1 Event encoding

The attribute layer (${}^nF_1^k$, n and k are the number of inputs and attributes) of Deep-ART contains detailed features of the input which are sent to the input field (F_2) through the input channels (${}^nI^k$). ${}^1F_1^1$ receives always an action v. For $n \neq 1$, ${}^nF_1^1$ takes a preposition and ${}^nF_1^k$ takes an object class o_n. An encoding example is shown in Table P4.1.
The input field (F_2) is used to encode the events themselves and works similarly as to aforementioned encoding procedure. The activation of the F_2 nodes is achieved by the choice function:

$$ {}^n_jT = \sum_{l=1}^{k} {}^n\gamma^l \frac{|\,{}^nx^l \wedge {}^n_jw^l\,|}{{}^n\alpha^k + |\,{}^n_jw^l\,|} \tag{P4.1} $$

where ${}^nx^k = [{}^nI^k, 1 - {}^nI^k]$ is the activity vector, ${}^n_jT$ is the activation value of the *j-th* node, ${}^n\gamma^l$ is a contribution parameter, ${}^n\alpha^k$ is a choice value and ${}^n_jw^l$ is a weight vector associated with the category j. The resonance value for the J-th node (the node with the highest activation value) is checked to be greater than a certain vigilance parameter (${}^n\rho^k$):

$$ {}^n_Jm^k = \frac{|\,{}^nx^k \wedge {}^n_Jw^k\,|}{|\,{}^nx^k\,|} \geq {}^n\rho^k; \text{where } J = \arg\max({}_jT) \tag{P4.2} $$

If this condition holds true, the weights are updated with the rule:

$$ {}^n_Jw^{k(new)} = (1 - {}^n\beta^k)\,{}^n_Jw^{k(old)} + {}^n\beta^k({}^nx^k \wedge {}^n_Jw^{k(old)}) \tag{P4.3} $$

where ${}^n\beta^k$ is a chosen learning rate. If the condition does not hold true, resonance is checked for the next largest activation value node. Eventually, the events will be categorized in the input channels (${}^iF_3$) of the event field F_3.

Table P4.1: Deep-ART encoding of subtask 6 in Table P4.2

POUR	*Wine_bottle*	in	*Wine_glass*
v ${}^1F_1^1$	o_1 ${}^2F_1^2$	γ ${}^3F_1^1$	o_2 ${}^3F_1^2$

2.1.2 Episode encoding

As mentioned above, the input channel ${}^{i}F_3$ categorizes events. Now, the buffer channel ${}^{b}F_3$ will save the values from an output channel vector ${}^{o}F_3$. Thus the episodes can be encoded such that:

$$ {}^{b}x_n = {}^{o}w\,{}^{o}y_{n-1} \tag{P4.4} $$

$$ {}^{o}y_n = {}^{i}w\,{}^{i}x_n + {}^{b}w\,{}^{b}x_n = {}^{i}w \sum_{k=0}^{n-1} ({}^{b}w\,{}^{o}w)^{k}\,{}^{i}x_{n-k} \tag{P4.5} $$

where ${}^{i}x_n$, ${}^{b}x_n$ and ${}^{o}y_n$ are the input, buffer and output vectors, respectively; and ${}^{i}w$, ${}^{b}w$ and ${}^{o}w$ their corresponding weights. Thus the output vectors memorize the event sequences which serve as inputs to the episode layer F_4. These episodes are then categorized in the input channel ${}^{i}F_4$. The encoding procedure can be repeated to obtain daily episodes F_5.

2.1.3 Episode retrieval

To retrieve episodes based on partial input cues, Deep-ART reverses the episode encoding process, as described in [1]. Thus, Deep-ART can recall past episodes (hence, task sequences) from similar current situations, just like a human would do.

2.2 Action sequence generator: FF-planner

Deep-ART can only retrieve episodes in prelearnt scenarios; if the environment changes, Deep-ART will fail to generate correct task sequences. Thus, the fast-forward planner (FF-planner) [8] is used to solve unlearnt tasks based on similar pre-learnt ones. The generated task sequences are transfered and stored back by Deep-ART as an episode. The FF-planner uses enforced hill climbing (EHC) as a search method and A* algorithm if EHC fails. It parses PDDL language [9] to represent a task planning problem as a tuple $\{A, s_0, s_g\}$, where A is a set of actions defined as preconditions and effects between objects and locations, s_0 is the starting state and s_g the goal state. Thus, the FF-planner will generate a new task sequence to achieve s_g based on combinations of pre-learnt behaviours.

2.3 Trajectory warping for new behaviours

The task sequences provided by Deep-Art and the FF-planner require certain behaviours that have been pre-trained, either by human demonstration or by moving the robots arm passively by an expert [10]. These actions depend on certain start and end states which will surely not match with those in real scenarios because object and end effectors position are variable (i.e. pouring, throwing, shaking...)
As proposed in [3], the end effector pose in a trajectory is represented as a twist vector $\xi = [\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T \in IR^6$, where ω is an angular vector and v a linear change. Thus the homogeneous transformation matrix between start ($_s$) and goal (^{g}) points will

be:

$$H_s^g = exp(\hat{\xi}_s^g) = exp\left(\begin{bmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{bmatrix}_s^g \right) \tag{P4.6}$$

The warping variation (H_{diff}) is obtained knowing the end-effectors start and end poses of the demonstrated trajectory and those of the actual environment ($H_o^s, H_o^g, \hat{H}_o^s, \hat{H}_o^g$, respectively), related to the origin frame ($_o$). Then, the warped twist vector $\hat{\xi}_{diff}$ is obtained:

$$H_{diff} = (H_o^s - H_o^g)^{-1} \cdot (\hat{H}_o^s - \hat{H}_o^g) \tag{P4.7}$$

$$\hat{\xi}_{diff} = log(H_{diff}) \tag{P4.8}$$

Finally, the end effector poses of the demonstrated trajectory are warped. H_s^k is the end-effectors pose at a *k-th* point in the demonstrated trajectory, N is the total number of these points and $\hat{H}_s^k$ are the end-effectors poses in the warped trajectory:

$$\hat{H}_s^k = \hat{H}_o^s H_s^k exp(k \frac{\hat{\xi}_{diff}}{N}) \tag{P4.9}$$

To follow these warped trajectory points whilst, simultaneously, avoiding collisions, the *OMPL MoveIt* implementation of the sampling based motion planning algorithm Quick-RRT* [11] is applied.

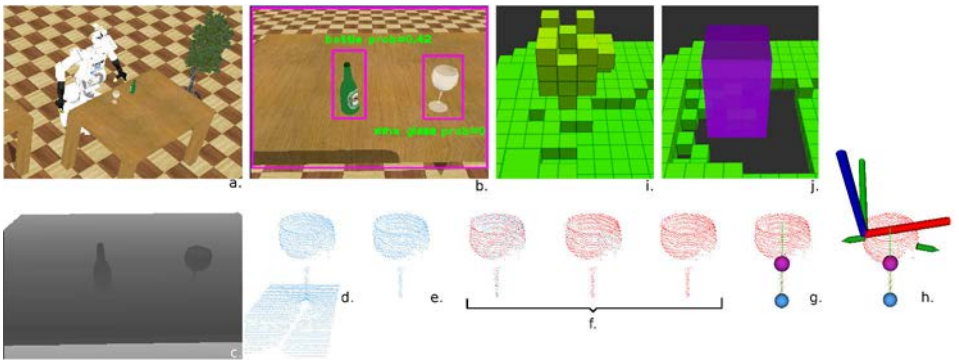


Figure P4.2: Perception pipeline for unknown objects: **a.** Simulation on *Webots* **b.** CNN results **c.** Rectified Depth image **d.** Point cloud (wineglass) **e.** Result of segmentation **f.** Registration (red) of new point clouds (blue) **g.** Data extraction: centroid (purple sphere), surface contact (blue sphere), principal direction (green vector) and overall size of point cloud **h.** Possible grasping poses (green arrows) and selection of best one (large axis) based on cosine similarity with closest hand pose **i.** Generated octomap **j.** Collision object based on extracted data that remove collision checking between robot and octomap.

3 Perception pipeline

The main contribution of this paper is adding an online object detection and segmentation system to the Task Intelligence problem. That is, adding the necessary perceptive means for the robot to understand a dynamic environment, memorize concepts and act accordingly.
Prior to this, the robot must evaluate whether the action or task that is to be carried out relies on the knowledge of a certain object. If so, the knowledge could be there already, from previous events or from manual human input. Nonetheless, if said object's information has not yet been found and memorized, the following section describes the perceptive procedure to do so. All in all, the procedure follows the pipeline presented by *Llopart et. al.* [12], with the difference that now only one object is being segmented allowing for faster processing times and better performance.

3.1 Detection

If no information of the object, included in the requested behavior, has not yet been stored in memory, it is necessary to find it first. Initially, if Deep-ART retrieves a pre-learnt position where the object is usually found, the robot will navigate to that location (i.e if the robot is requested to grab a knife, these are commonly found on the kitchen table). Once the position is reached, the robot will *look around* its environment until it finds the object using an RGB-D sensor mounted on the head (Fig. P4.2a.). The robot will turn its head 30° both pitch and yaw-wise with a 3° step. It will evaluate if the object has been detected at every step.
Object detection, recognition and semantic labelling is done using a Convolutional Neural Network (CNN) through which Regions of Interest (ROI) are extracted in real time around a labeled class (Fig. P4.2b.). The YOLO Real-Time Object Detection System ([13]) is employed since it has been pre-trained on the COCO dataset to detect up to 80 different classes, which include, amongst others, common household items and kitchenware. Then, the obtained bounding box is converted into a binary mask which is used to crop the obtained rectified depth images (Fig. P4.2c.). By converting this data to a point cloud, only the points that are within the found ROI will be accepted (Fig. P4.2d.). These will represent mainly the desired object plus some background and noise. The centroid of the point cloud is then found (which represents a vague estimate of the objects position) and stored in memory (inside an *object's class*) with the object's semantic label found by the CNN. This way, the information can be later referenced if the object has to be found again; this positional data can be updated online if a more precise value is found or if the object is moved.

3.2 Segmentation and registration

Once an estimated position of the object is found, and the robot is correctly facing it, the segmentation process takes place. Partial point clouds representing the object are determined continuously but they include noise and background information which needs to be removed to obtain an accurate model of the object.
Initially, all point clouds are filtered to remove encoding errors (NaN) and noise, thus an outlier removal filter (with a radius of 10 cm and a minimum point threshold of 4)

is applied. Additionally, a pass through depth filter is also employed to remove those points closer than 20 cm to the sensor (they are probably noise too) and those further than 5 meters since they lack precision.
Up to 7 segmentation methods are applied based on the semantic label previously found by the CNN, as described in [12], to remove unwanted data (Fig. P4.2e.). This means that depending on the object detected, one or another segmentation method is selected. These remove those points that do not represent the object (such as tables or walls). Moreover, some of these methods derive additional information about the object. For instance, cylindric segmentation gives an estimate of the radius and height of an object (used for bottles or cups), whilst spherical segmentation only provides the radius (used for apples or oranges). These values are then also stored inside the *object's class*.
A more reliable model of the object can be built by registering multiple object's point clouds over a small amount of time (Fig. P4.2f.). This procedure is accomplished by applying the Iterative Closest Point (ICP) between two sets of point clouds, and, if it converges, merge the transform of one point cloud with the other. The ICP algorithm minimizes the point coordinates between two sets of point clouds to find the best data transform between them; this allows to take the continuously generated point clouds and add them up together, if they are close enough already, resulting in the best possible model of the object. Plenty of information can be then extracted from these models.

3.3 Extracting data

To be able to interact with and manipulate objects, some data must be obtained from the objects models (Fig. P4.2g.). This knowledge is updated in the *object's class* that is being stored in memory.
A better centroid estimation of the object is found; hence, the object's initial estimated position is updated. The normalized covariance matrix and the eigenvectors that represent the principal axes of the point cloud are derived to find the object's orientation. After transforming the point cloud to the new reference frame, the maximum and minimum point values in each direction are obtained. This provides significant insight into the objects dimensions and allows the extraction of values such as maximum height, point of contact with surface (i.e in the cases where the objects are on top of tables) and, most importantly, serves to delimit the point clouds dimensions.

3.4 Collision object

The dimensionality information is also stored in memory because, with it, a *collision object* can be created (Fig. P4.2j.). These are built using a rectangular prism, cylinders or spheres, based on the labeled semantics of the object. This means that those objects that have been segmented using cylindrical or spherical methods will require a cylindrical or spherical *collision object*, respectively; whilst the rest will be modeled as a rectangular prism for simplicity.
The reason for constructing such *collision objects* is that it enables a robot running the *MoveIt* libraries from ROS to discern between objects and a generated octomap (Fig. P4.2i.). The octomap is a 3D occupancy grid mapping method that models the robots surroundings as occupied, free or unknown space, which helps avoid collisions when navigating or moving around. However, objects will always be included inside the occupied space which will deny the robot any possible interaction with them. Therefore,

by adding *collision objects* to the octomap, the collision checking between robot and objects is disabled, allowing said interaction.

3.5 Grasp pose selection

Selecting a correct grasping pose is a very relevant step prior to any sort of interaction between robot and object. The *agile_grasp* [14] package requires only a pointcloud (geometry of the objects model) and the robotic grippers dimensions to be able to sample grasp hypothesis which are then validated, if the grasping points are antipodal, through a Support Vector Machine (SVM). This procedure results in an array of grasping poses (represented as green arrows in Fig. P4.2h.) from which the best one must be selected.
The best grasping pose within the extracted array of poses has to be selected. This is attained by finding the current position of both arms end effectors and deriving the Euclidean distance from each of them to the objects centroid (calculated before). This metric helps know which hand is closest to most grasp poses, making the path planning easiest.
The cosine similarity measure ($cos(\theta)$) is calculated for all the elements inside the grasping poses array to select the best pose candidate. This value measures the similarity between two non-zero vectors by calculating the cosine of the angle between them. Therefore, it evaluates orientation and not magnitude: having a cosine similarity of 1 means both vectors have the same orientation, if the result is 0 the vectors are orthogonal, and if the cosine similarity is -1 the vectors are diametrically opposed.

$$cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{P4.10}$$

Where A_i and B_i are components of a vector A and vector B, respectively. Vector A describes a vector parallel to the ground and the robot (thus following the X reference axis) and its direction is determined by the hand closest to the object, which was selected before: if the right arm was selected, vector A has a value of *(-1, 0, 0)*, if the left arm is selected, the vector takes the value *(1, 0, 0)*. Vector B is the orientation of each grasp pose in the array. The grasp pose with the highest cosine similarity (larger than 0 and closest to 1) is the chosen one.
However, sometimes, due possible malformations of the grasping poses in which their direction might be opposite from the closest arm, all cosine similarity results will be smaller than 0. In this special case, a new grasping pose must be derived. To achieve this, the cosine similarity procedure is repeated but using the reversed grasp orientation vectors and the centroid of the object. All in all, a final correct grasp pose is selected based on the closest arm to the object, the objects geometry and the hands dimensions.

4 Experiments

The proposed perceptive pipeline for Task Intelligence system has been tested on a simulated environment to evaluate its performance and effectiveness. The humanoid robot *Mybot* was modelled and used in this simulation.

4.1 Mybot Humanoid Robot

The Robot Intelligence and Technology Laboratory (RIT) at KAIST has developed a wheel-based humanoid robot, *Mybot*. It uses an Odroid XU board with Ubuntu 16.04 and ROS Kinetic Kame. The robot incorporates a total of 26 Degrees of Freedom (DOF): 8 DOF for each arm and 3 per hand, with 2 additional DOF in the waist (tilt and pan). The mobile base is omni-directional for easy navigation and contains a 2D laser rangefinder. The head, containing an Xtion ASUS sensor used to obtain rectified RGB and Depth images of the surroundings, has 2 DOF (pitch and yaw).

4.2 Experimental Setup

All tests have been run inside the *Webots* simulator; in the future, the real robot will be used. A desktop computer with a GTX 1080 GPU card and running Ubuntu 16.04 and ROS Kinetic Kame is employed for such simulations. A scenario with two tables is reproduced, where each table has a series of objects on it. The Deep-ART has pre-learnt a series of episodes using PDDL (the ones from [3]) and an additional one which includes the task sequence of pouring wine from a bottle onto a glass. The exact locations, geometry and grasps of the objects are initially unknown since finding those values is the main objective of the proposed approach.

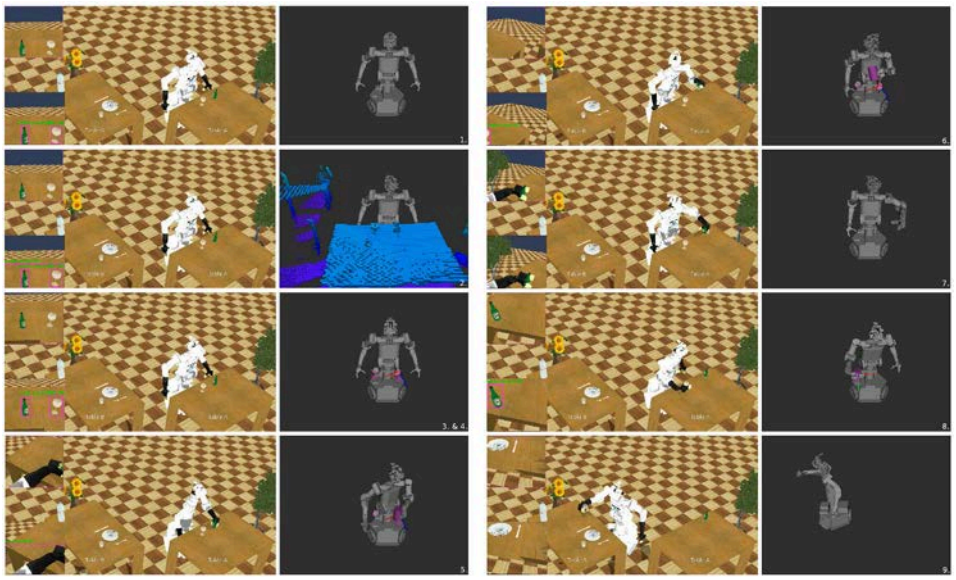


Figure P4.3: The images depict the behaviours retrieved by Deep-ART for the *Bring wine to Table_B* Task. Their actions and order are shown in Table P4.2. **Left.** *WeBots* simulator illustrating the current view of the robot (upper image) and the results provided by the CNN (lower image) **Right.** RViz visualizer showing object segmentation and data extraction.

Table P4.2: TASK: Bring cup of wine to Table_B

1	MOVE to *Table_A*
2	CREATE *octomap*
3	FIND *Wine_glass*
4	FIND *Wine_bottle*
5	GRASP *Wine_bottle*
6	POUR *Wine_bottle* in *Wine_glass*
7	RELEASE *Wine_bottle* on Table_A
8	GRASP *Wine_glass*
9	MOVE to Table_B
10	RELEASE *Wine_glass* on Table_B

4.3 Experimental Results and Discussions

When the robot is asked to "bring a cup of wine to table B", Deep-ART inferred a task sequence from the partial cue. Specific actions/behaviours were also retrieved by the FF-planner because they had not been learnt beforehand.
Initially, the robot would move to Table A where the wine bottle and glass were supposed to be on. To travel there, an octomap of the environment must be generated which is updated online. This allows collision avoidance for all other actions of the robot in the future. Once the robot is in front of Table A, it must look around to find precisely where the required objects are. The procedure has been described in Section 3 and results in the detection of the specified objects and their correct labeling, the precise estimate of the object's position on top of the table, their centroid, geometry and dimensions, and the best grasping pose available based on the closest hand to them.
In this case, the wine bottle was grasped and, knowing the position of the ring/top neck of the bottle, its grasping pose and the centroid of the wine glass, a previously learnt *pouring* behaviour is warped, as seen in Section 2.3, to match these initial and final states. It is worth mentioning that if the grasping of an object with one hand or its motion planning fail, the same pipeline takes place but using the other hand to be able to try all possible cases before the robot gives up. Once the pouring action ends, the wine bottle is released again on top of the table and the (now full) wineglass is grasped and taken to Table B where it is released, ending the whole episode. This procedure is described in Table P4.2 adn seen in Fig. P4.3, where the locations and objects are written in *italic* and the behaviours in *capitals*.

5 Conclusion

In this paper, an additional module for the Task Intelligence problem is proposed. The previous system works in constrained environments where most of the object's information was pre-defined. This system is formed by three modules: the Episodic Memory (Deep-ART) retrieves a task sequence after a certain partial input cue. The action sequence generator (FF-planner) allows the creation of new task sequences to solve unlearnt cases. Finally, the trajectory warping module is used to perform specific behaviours for each task based on the robot's and object's current states.
The presented perceptual pipeline builds on and enhances the preceding system, allowing the robot to understand and interact with a dynamic environment whilst updating

online all information on the surrounding objects. Thus, this more generalized approach requires considerably less prior knowledge of the environment and objects than previous approaches but equips the robot with the ability to learn about its surroundings dynamically, memorize the new data and use it to its advantage. This is achieved using a CNN to detect and recognize objects, various segmentation methods, a selection of best grasping poses and a XMl parser to be able to share or update the newly registered data with the other Task Intelligence modules.
The results of the simulated experiment show a robot that, given a partial instruction, retrieves the necessary task sequence from previously learnt episodes (memories). It uses the newly presented perceptual pipeline to comprehend and extract data of its environment. It then follows the retrieved task sequence, modifying its behaviours based on the objects data, until it finally accomplishes its goal.

6 Future Work

The proposed pipeline works as intended in the presented test case. However, further scenarios must be developed, not only in simulation, but also in the real world. Having a wider range of demonstrated trajectories (actions) and a larger number of recognizable classes by the CNN, would allow for more complex and interesting task sequences. Novel neural networks, like Mask-RCNN (proposed by *He et. al.* [15]) result in the pixel-wise semantic segmentation of objects. The usage of networks like these, even at the cost of a lower processing speeds, allows for the elimination of the segmentation step in the 3D point cloud domain (as described in Section 3.2). Finally, adding a multi-viewpoint perspective ([16] and [17]) when registering the objects point clouds would entail more reliable objects models, thus more accurate extracted data and better grasping poses.

Bibliography

[1] G.-M. Park, Y.-H. Yoo, D.-H. Kim, and J.-H. Kim, "Deep art neural model for biologically inspired episodic memory and its application to task performance of robots," *IEEE transactions on cybernetics*, vol. 48, no. 6, pp. 1786–1799, 2018.

[2] I.-B. Jeong, W.-R. Ko, G.-M. Park, D.-H. Kim, Y.-H. Yoo, and J.-H. Kim, "Task intelligence of robots: Neural model-based mechanism of thought and online motion planning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 41–50, 2017.

[3] D.-H. Kim, G.-M. Park, Y.-H. Yoo, S.-J. Ryu, I.-B. Jeong, and J.-H. Kim, "Realization of task intelligence for service robots in an unstructured environment," *Annual Reviews in Control*, 2017.

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

[5] A.-H. Tan, G. A. Carpenter, and S. Grossberg, "Intelligence through interaction: Towards a unified theory for learning," in *International Symposium on Neural Networks*. Springer, 2007, pp. 1094–1103.

[6] W. Wang, B. Subagdja, A.-H. Tan, and J. A. Starzyk, "Neural modeling of episodic memory: Encoding, retrieval, and forgetting," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 10, pp. 1574–1586, 2012.

[7] G. M. Park, Y. H. Yoo, and J. H. Kim, "Rem-art: Reward-based electromagnetic adaptive resonance theory," in *Proceedings International Conference on Artificial Intelligence (ICAI)*, 2015, pp. 805–811.

[8] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 639–646.

[10] A. Jain, S. Sharma, T. Joachims, and A. Saxena, "Learning preferences for manipulation tasks from online coactive feedback," *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1296–1313, 2015.

[11] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Rrt*-quick: A motion planning algorithm with faster convergence rate," in *Robot Intelligence Technology and Applications 3*. Springer, 2015, pp. 67–76.

[12] A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Generalized framework for the parallel semantic segmentation of multiple objects and posterior manipulation," in *Robotics and Biomimetics (ROBIO), 2017 IEEE International Conference on*. IEEE, 2017, pp. 561–568.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[14] A. ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," in *Robotics Research*. Springer, 2018, pp. 307–324.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2980–2988.

[16] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4628–4635.

[17] C. Li, H. Xiao, K. Tateno, F. Tombari, N. Navab, and G. D. Hager, "Incremental scene understanding on dense slam," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 574–581.

Publication P5

Semantic mapping and object detection for indoor mobile robots

Szymon Kowalewski[1], Adrian Llopart[1] and Jens Christian Andersen[1]

Abstract

In this paper the authors present a full solution for object-level semantic perception of the environment by indoor mobile robots. The proposed solution not only provides means for semantic mapping but also division of the environment into clusters representing singular object instances. The robot is provided with information that not only allows it to avoid collisions with obstacles present in the environment, but also information about the localization, the class and the shape of each encountered object instance. This level of perception enhances the robot's ability to interact with the environment.

The state-of-the-art deep learning solution, Mask-RCNN, is used for the image segmentation task. The image processing network is combined with an RTAB-Map SLAM algorithm to generate semantic point clouds of the environment. The final part of the paper is focused on point cloud processing: providing methods for instance extraction and instance processing.

To verify the performance of the proposed methodology multiple experiments are conducted. Through the evaluation of the results it is possible to identify possible improvements.

Keywords: deep learning, Mask-RCNN, neural networks, image segmentation, point cloud segmentation, obstacle detection, robotics

1 Introduction

The environmental awareness is a crucial feature of an autonomously operating robot. The information about surroundings is acquired by available sensors and then processed internally to make autonomous operation in an unknown environment possible. The most basic approach is to use sensors to detect occupied areas and mark them as obstacles. In such case, the robot is aware of the placement of obstacles but lacks any information about the type of the detected objects. This approach is sufficient when

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **szymon.kowalewski@gmail.com, (adllo, jca)@elektro.dtu.dk**

the objective is to maneuver and avoid collisions but doesn't provide any additional information, that might be useful when interacting with encountered objects.

The methodology proposed in the paper expands this basic approach by adding an obstacle classification step and instance detection step. In the result, not only the occupancy information is acquired but also the semantic classification for each detected object is determined. Such approach allows to not only navigate around objects but also interact with them depending on the obstacle class; for instance a chair might be moved, whereas a table should be maneuvered around.

The main contributions of this paper are as follows:

- Design of an end-to-end solution that provides an object-level classification and outputs class, position and shape of each of the detected objects.
- Verification of the performance through experiments.

The rest of the paper is divided in following manner: in Section 2 related work is reviewed. Section 3 provides the reader with a general overview of the proposed solution and the division of the problem into segments. In sections 4, 5 and 6 the methodology used in each of the segments is described. Section 7 includes the description of conducted experiments. The results of experiments are presented in Section 8. Sections 9 and 10 include conclusion and future work, respectively.

2 State of the art

There's been multiple attempts to provide solutions capable of generating 3D semantic maps - maps in which each area is semantically classified. The most common methodology combines a semantic image segmentation solution with a SLAM algorithm.

An example of this approach is SemanticFusion [1] which is a combination of, at the time, the state-of-the-art Deconvolutional Semantic Segmentation network [2] with an ElasticFusion SLAM [3] algorithm and a Bayesian update scheme. This approach allows authors to generate accurate semantic 3D maps of the environment. Nakajima et al. [4] take a similar approach but they assign class probabilities to regions instead of single voxels of the 3D map. This provides a significantly faster solution that can perform at 30Hz frequency. Jeong et al. [5] present a methodology based on the same approach but use multiple sensors to allow outdoor, large scale operation.

A slightly different approach to the segmentation problem is presented by the authors of PointNet [6]. PointNet is a deep neural network that provides end-to-end point cloud processing capabilities, outputting a variety of results including: object classification, part segmentation and semantic segmentation. Ability to directly process point clouds is believed to provide superior results. The presented solution is used for single frame processing and authors do not explore the idea of generating global maps.

Nonetheless, these works are focused on the generation of the semantic map and do not attempt the object-level instance division as proposed in this paper.

3 Overview

The proposed system was divided into four essential sub-modules:

Image acquisition - this module is responsible for the acquisition of RGB and depth images from the Kinect camera.

Image segmentation - this module is responsible for processing acquired images. The performance of this module has a great impact on the whole system as this module adds the semantic meaning to the acquired images. The output images include annotations about detected objects, their classes and localization in the image. Mask-RCNN [7] is used to achieve the module's functionality. For the purpose of the neural network training and performing experiments a GeForce GTX 1080 graphical unit was used.

Robot localization and mapping module - this module is responsible for keeping track of the robot position and simultaneously creating a semantic point cloud of the environment. RTAB-Map [8] is used to achieve the module's functionality.

Instance detection and processing module - this module is responsible for splitting the created semantic point cloud into separate object instances. The output of this module is the localization (in the global coordinate system), class and the shape (represented by a point cloud) of each of detected objects. Moreover, this module is responsible for the instance processing task.

4 Image segmentation

To achieve the semantic image segmentation the Mask-RCNN deep neural network is used. The Mask-RCNN is a neural network architecture created in Facebook AI Research, it expands the capabilities of it's predecessor: Faster-RCNN[9], by adding an image segmentation output. The network processes the input image and provides localization of the classified objects together with segmentation masks.

The original RGB version of Mask-RCNN is modified to process RGB-D images. The reasoning behind the modification is that in many cases the objects seem to be hard to distinguish using only RGB images (for instance when colors of the object and background are similar). In these cases the depth image might provide the missing information necessary to detect object's boundaries and therefore enhance the performance of the network.

The neural network was trained using the SUN-RGBD dataset[10]. This dataset is a compilation of multiple different indoor datasets [11][12][13] and consists of 10.335 images. It contains 37 different object classes which represent objects that are expected to be found in an indoor environment. The dataset provides comprehensive labeling including all of annotations required for Mask-RCNN training with a sub-set of classes (chair, table, window, door, box, shelves, sofa, cabinet). The dataset was split into training and validation sets as provided in the dataset information file.

The pre-trained model used for transfer learning was a Mask-RCNN model trained on the the COCO dataset [14]. Multiple training sessions were conducted using RGB and RGB-D variations of Mask-RCNN. For the training sessions in which the Mask-RCNN enhanced with depth was used, the weights of the first convolutional layer of the network could not be transferred as the shape of the convolution was modified. In such cases, this layer was ignored during the transfer learning process.

The results and discussion can be found in Section 8.1.

5 Robot localization and mapping

To be able to map the environment, as the robot is exploring the environment, it is necessary to reliably localize the robot. Moreover, the semantic images generated by the image segmentation module have to be expanded into a point cloud using the corresponding depth images. These actions are achieved by applying a simultaneous localization and mapping (SLAM) algorithm.

RTAB-Map algorithm was found to be suitable for the task. It is an RGB-D Graph-Based SLAM algorithm that ensures real-time operation. The robot localization and mapping module outputs a semantic 3D point cloud of the environment together with a 2D cost-map of inaccessible areas.

The original implementation of RTAB-Map algorithm had to be adjusted to be suitable for the solution proposed in this paper. A parallel graph was added that includes segmented images instead of the original RGB frames captured during the operation. This approach allows to keep the original operation of the RTAB-Map, which is based on the features extracted from camera frames, and at the same time generate a point cloud based on semantic images.

6 Instance detection and processing

6.1 Instance detection

The generated semantic point cloud provides substantial amount of information about the environment but lacks the division into single object instances. To achieve this task, the generated semantic point cloud of the environment is processed by an euclidean cluster extraction algorithm. The parameters used for the cluster extraction can be found in Table P5.1.

To avoid clustering together points belonging to different classes the full semantic point cloud is divided into subsets corresponding to each of the classes. The euclidean cluster extraction is applied to each of the subsets.

6.2 Instance processing

The final part of the solution consists of a set of instance processing algorithms. Instance processing is a crucial part of the proposed solution, it illustrates the usefulness of the approach and possible capabilities of a robot with object-level perception of the environment. Two instance processing examples were implemented: chair orientation detection algorithm and approach pose estimation algorithm.

The chair orientation detection algorithm allows the robot to determine the orientation of each of the encountered chair objects. This is beneficial when interacting

Table P5.1: Euclidean cluster extraction parameters

Parameter	Value
Search radius	10[cm]
Minimum size	50[points]

with chair objects as the robot is aware from which direction the object should be approached.

The approach pose estimation algorithm is used to, given the selected object instance, estimate the target position of the robot when navigating towards the object.

6.2.1 Chair orientation detection algorithm

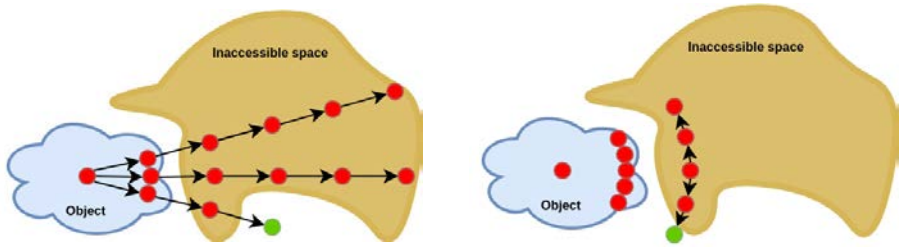
The algorithm for detection of the front of the chair is divided into two parts. Firstly, a plane parallel to the z axis is fitted into the instance point cloud. To find the plane the RANSAC algorithm is used. The detected plane corresponds to the backseat of a chair. Obviously this part works only with the assumption that the chair has a distinguishable backseat. The plane provides two normal directions that represent the front and the back of the chair.

Once the backseat plane is detected it is necessary to determine which direction represents the front and which represents the back of the chair. This part of the algorithm works on the assumption that distance-wise, the front of the chair is longer than the back. To determine the orientation of the chair a simple algorithm was proposed: points with maximum distance from the plane are found on the both sides on the plane; the one with a larger distance is determined to be on the front side of the plane.

The detected front direction is then represented with a unit vector placed in the mean point of the instance point cloud. To allow operations on the 2D obstacle map, the found vector is casted onto the ground plane.

6.2.2 Approach pose estimation algorithm

The approach pose estimation algorithm was introduced to add a capability of navigating the robot towards a selected object instance. To achieve this, the space around the object is sampled by checking in the general obstacle map if the location is accessible. The space is sampled either in ray-like or arc-like fashion. The operation of the algorithm is illustrated in Fig. P5.1.



(a) Ray sampling

(b) Arc sampling

Figure P5.1: Approach pose estimation algorithm. Red dots represent unsuccessful sampling steps whereas green dots represent the final successful attempt.

7 Benchmarking methods and experiments

7.1 Guidebot

To be able to perform experiments in the real-life scenario and verify the performance of the system it was necessary to use a mobile robot. For this purpose a DTU robot - Guidebot was used. It is a differential drive wheeled mobile robot equipped with a range of sensors including: a Kinect camera (XBOX 360), a laser scanner (SICK LMS200-30106) and wheel encoders.

7.2 Image segmentation

Multiple training sessions were run to provide more insight into the optimal training procedure. To measure the performance of the image segmentation task the mean average precision (mAP@0.5) [15] is used. The results are presented in Section 8.1.

7.3 Full solution

A final experiment was proposed to verify the performance of the whole solution in a real life scenario. The experiment consisted of two stages that are described in detail in this section. A single room was setup by placing different recognizable objects at known positions. On Figure P5.3a it can be seen that to make the task more challenging, a variation of objects was selected for each class (for example round and rectangular tables or different types of chairs). Table P5.2 contains information about objects used in the experiment and their placement. Additionally the last column provides information if the object was detected by the algorithm during the experiment.

During the experiment it was noticed that the robot detects background objects. The localization measurement was not conducted for these objects but for the sake of the completeness of the experiment they are present in the Table P5.2 with "unknown" coordinates.

During the first stage of the experiment the robot was manually controlled. The goal was to map the robot's environment and create a global semantic point cloud. Then, the instance detection algorithm provided information about the position and the number of detected objects. These results allow to quantify the performance of the robot by verifying how many of the objects were detected and how precise the solution is.

For the detected objects, a mean localization error was calculated. The error was derived using the following formula, where $\boldsymbol{d}$ is the number of the detected objects, x_i^{gt} and y_i^{gt} are the coordinates of the i-th ground truth object, x_i^{pred} and y_i^{pred} are the coordinates of the corresponding predicted object.

$$e = \frac{\sum_{i=1}^{d} [abs(x_i^{gt} - x_i^{pred}) + abs(y_i^{gt} - y_i^{pred})]}{d} \tag{P5.1}$$

Thus, before the start of the second stage of the experiment, the robot has already mapped the environment and is aware of detected objects. Commands were sent to the robot to approach multiple of the detected objects to see if the robot was able to maneuver in the environment and reach the selected objects.

Table P5.2: Placement of the objects during the final experiment

Object	x[m]	y[m]	Detected?
Chair #1	1.25	-0.85	Detected
Chair #2	2.10	1	Detected
Chair #3	3.10	1	Detected
Table #1	2.55	1	Detected
Table #2	0.4	1.2	Detected
Box #1	0.3	1.2	Detected
Box #2	4.45	-0.55	Detected
Box #3	unknown	unknown	Detected
Box #4	unknown	unknown	Detected
Box #5	unknown	unknown	Detected
Shelves #1	3	-0.85	Not detected
Shelves #2	unknown	unknown	Detected
Shelves #3	unknown	unknown	Detected

8 Results

8.1 Segmentation results

To find a model with an acceptable performance, the training sessions were conducted using both: the RGB architecture and the RGB-D architecture. The compilation of performed training sessions can be seen in Fig. P5.2.

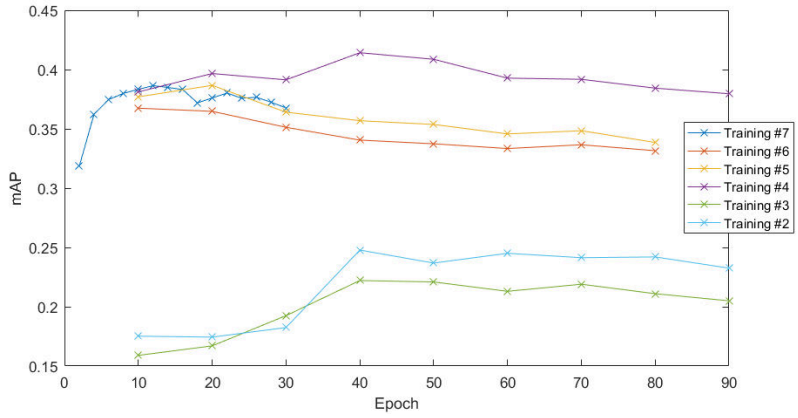


Figure P5.2: Performed training sessions. During the training sessions #2 and #3 the initial convolutional layers were frozen and therefore the performance of these models is inferior. After noticing that the models tend to overfit during long training procedures, session #7 was ran shorter with a finer performance sampling.

Table P5.3: mAP of the best model for each of the classes. It can be observed that classes with more training instances have performed better than classes underrepresented in the dataset.

Class	**mAP**	Number of training instances
Chair	0.631	11514
Table	0.53	6913
Window	0.464	2549
Door	0.415	2000
Cabinet	0.399	1464
Sofa	0.334	701
Box	0.229	1982
Shelves	0.203	482

The best performing model achieved the mAP of 0.414 (perfect mAP score is 1). The visual inspection of the results has proven a good performance of the network. The model was trained for 90 epochs (450000 iterations). SGD optimization was used with a learning rate of 0.001. The architecture of the best performing model was designed for RGB images as input. A more detailed analysis has been performed for the best performing model: Tab. P5.3 shows the performance of the model and number of training examples for each of the classes.

Unexpectedly, the Mask-RCNN with RGB-D inputs has achieved inferior results (maximum mAP 0.386). The reason for this might be the fact that the first convolutional layer was ignored during the transfer learning procedure. Moreover, for each hyper-parameters setting only a single training session was conducted. Due to the stochastic nature of neural networks a single training result does not provide a definite answer about performance differences.

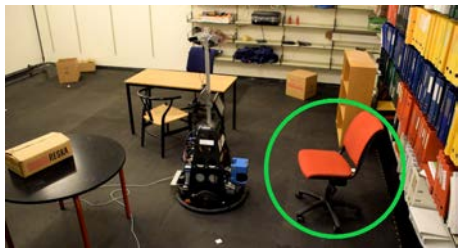
8.2 Full solution results

Firstly, it was observed that the final solution achieved a high recall value - detecting nearly all of the present objects. The recall value for the class, table and box classes reaches 1.0 what means that all of the ground truth objects were detected. For the shelf class the recall value drops to 0.67 due to the fact that one of the shelves was not detected.

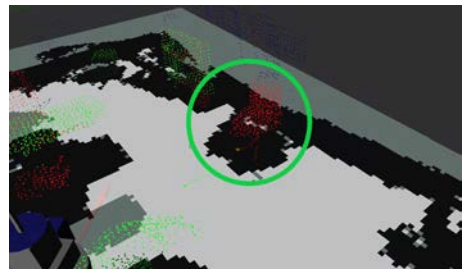
The robot performs noticeably worse when the precision value is taken into account. This means that the final solution detects multiple objects that were not present in the environment. The robot detected 1 chair, 3 tables and 2 windows that were not present in the prepared room. It is worth noticing point clouds for the majority of the miss-classified instances consist of small number of points. This means that the solution might be improved by changing the point cloud acceptance threshold in the instance detector.

The calculated mean localization error value (see Formula P5.1) is 0.25[m]. It is expected that even better results could be achieved if the mapping process was extended to allow the robot to see the object from increased number of viewpoints.

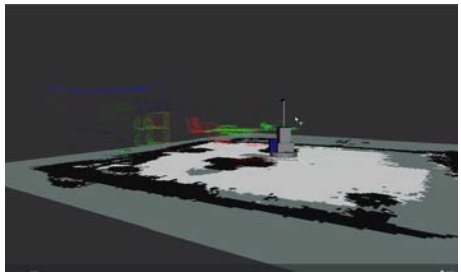
For the second stage of the experiment, three detected object instances from the following classes were chosen: a chair, a table, a box. For each of the tested object the



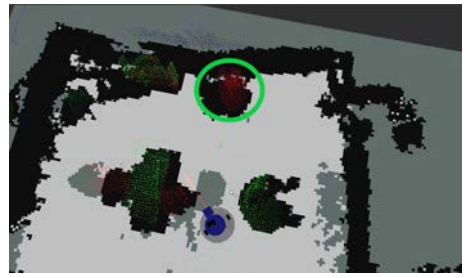
(a) The experiment environment



(b) Semantic point cloud viewpoint A



(c) Semantic point cloud viewpoint B



(d) Semantic point cloud viewpoint C

Figure P5.3: The experiment environment and the generated semantic point cloud seen from different viewpoints. Light green - chair class; Dark green - box class; Red points - table class; Blue - shelves class. To make scene interpretation easier for the reader, one of the chairs is marked with a green circle.

solution provided positions and orientations that would result in the robot facing the selected obstacle. The determined pose was visualized on the screen and was correct. Using a Robot Operating System (ROS) navigation package it was possible for the robot to avoid collisions and approach the selected object, given a desired end pose.

9 Conclusion

The main objective of the paper was to provide an semantic mapping and instance detection solution for indoor mobile robots.

The methodology has been designed, implemented and verified. The final experiment has shown that the robot equipped with the proposed system was able to map it's environment and assign semantic meaning to sections of the map. Moreover, the robot was able to divide the environment into single instances of objects belonging to various classes - keeping track of their localization and shape. Additionally, it has been proven that the acquired information about the shape of objects can be reliably used to determine orientation of objects. Finally, it was possible to use the acquired information to autonomously navigate the environment and approach objects.

To improve the results of the system, a detailed analysis of the performance of the image segmentation module has been conducted. A modification to the RGB Mask-RCNN implementation has been introduced by adding a depth channel to the input. No improvement of the performance was observed for the network with RGB-D inputs.

Table P5.4: Detected objects and their placement. Green: true positives; Red: false positives; Blue: detection that were present in the environment but not considered during the setup of the experiment

Object	x[m]	y[m]	Notes
Chair #1	1.2	-0.95	Matched
Chair #2	3.18	1.13	Matched
Chair #3	1.94	1.21	Matched
Chair #4	2.57	-0.98	Shelves missclassified as a chair
Table #1	2.36	0.94	Matched
Table #2	0.57	1.18	Matched
Table #3	3.17	-1.04	Shelves missclassified as a table
Table #4	5.13	1.05	Missclassification
Table #5	4.98	0.81	Missclassification
Window #1	5	1	Missclassification
Window #2	1.29	-1.46	Missclassification
Box #1	0.33	1.31	Matched
Box #2	4.87	0.4	Detected a background box
Box #3	4.27	-0.3	Matched
Box #4	5.22	0.63	Detected a background box
Box #5	4.95	1.81	Detected a background box
Shelves #1	1.62	-1.47	Detected background shelves
Shelves #2	4.96	0.35	Detected background shelves

10 Future work

Although the results of experiments are satisfying, there are possibilities to further improve the performance of the solution. A more insightful Mask-RCNN training procedure should be conducted to provide a definite answer about the performance of the RGB and RGB-D Mask-RCNNs. Additional training sessions should be run for original and modified versions of the network to remove the impact of performance variance. Moreover, the performance of a neural network depends strongly on the number of available training examples. Throughout experiments it was proven that many of the classes available in the used dataset were underrepresented and hence the performance for these classes was inferior. A different dataset might be used or the existing dataset might be expanded to improve the quality of the image segmentation process.

The used SLAM algorithm (RTAB-Map) adds a significant limitation to the solution - the environment has to be static. In the future, it would be reasonable to replace the current SLAM algorithm with one that is suitable for operation in a dynamic environment. This would greatly expand capabilities of the solution.

Only a few scenarios for instance processing were presented. It is believed, that the paper provides a foundation to implement various instance processing algorithms

- allowing the robot to approach and interact with different classes of objects. In the future, a set of behaviours should be implemented for each of the detectable object classes.

Bibliography

[1] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *CoRR*, vol. abs/1609.05130, 2016. [Online]. Available: http://arxiv.org/abs/1609.05130

[2] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015. [Online]. Available: http://arxiv.org/abs/1505.04366

[3] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016. [Online]. Available: https://doi.org/10.1177/0278364916669237

[4] Y. Nakajima, K. Tateno, F. Tombari, and H. Saito, "Fast and accurate semantic mapping through geometric-based incremental segmentation," *CoRR*, vol. abs/1803.02784, 2018. [Online]. Available: http://arxiv.org/abs/1803.02784

[5] J. Jeong, T. S. Yoon, and J. B. Park, "Multimodal sensor-based semantic 3d mapping for a large-scale environment," *CoRR*, vol. abs/1802.10271, 2018. [Online]. Available: http://arxiv.org/abs/1802.10271

[6] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016. [Online]. Available: http://arxiv.org/abs/1612.00593

[7] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[8] M. Labbé. (2018) Real-time appearance-based mapping. [Online]. Available: http://introlab.github.io/rtabmap/

[9] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[10] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[11] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[12] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell, "A category-level 3-d object dataset: Putting the kinect to work," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Nov 2011, pp. 1168–1174.

[13] J. Xiao, A. Owens, and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 1625–1632.

[14] Matterport. (2018) Mask-rcnn model pre-trained on coco dataset. [Online]. Available: https://github.com/matterport/Mask_RCNN/releases\/download/v2.0/mask_rcnn_coco.h5

[15] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11263-009-0275-4

[16] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[18] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015. [Online]. Available: http://arxiv.org/abs/1511.00561

[19] J. Redmon, "Yolo: Real-time object detection website," https://pjreddie.com/darknet/yolo/, accessed: 2018-06-05.

[20] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez, "A review on deep learning techniques applied to semantic segmentation," *CoRR*, vol. abs/1704.06857, 2017. [Online]. Available: http://arxiv.org/abs/1704.06857

[21] A. Maurin, O. Ravn, and N. Andersen, *Door and cabinet recognition using convolutional neural nets and real-time method fusion for handle detection and grasping*. IEEE, 2017, pp. 144–9.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[24] Matterport. (2018) Mask-rcnn implementation. [Online]. Available: https://github.com/matterport/Mask_RCNN/

[25] NYU. (2018) Nyu dataset image example. [Online]. Available: https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html

[26] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[27] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[28] A. Concha and J. Civera, "Dense Piecewise Planar Tracking and Mapping from a Monocular Sequence," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[29] M. Sokolov, O. Bulichev, and I. Afanasyev, "Analysis of ros-based visual and lidar odometry for a teleoperated crawler-type robot in indoor environment," 07 2017.

Publication P6

Outlook for Navigation - Comparing Human Performance with Robotic Solution

Morgens Blanke[1], Søren Hansen[1], Adrian Llopart[1], Jonathan Dyssel Stets[2], Thomas Køster[3] , Jesper E. Brøsted[3] , Nicolai Nykvist[1] , Jakob Bang[4]

Abstract

Considering whether a temporarily unattended bridge could be allowed, Maritime Authorities wish to investigate whether sensor technology is available that, when seconded by sophisticated computer algorithms, is able to provide outlook with the same reliability and safety as that of the average human outlook. This paper reports findings from a comparative study of human versus electronic outlook. Assessment of navigator's outlook is based on measurements with a wearable eye-tracker and areas of their visual attention are recorded on video. Simultaneously, a set of electro-optical sensors provides image-data as input to computer algorithms that detect and classify objects at sea within visual range. Both the navigator and the computer algorithms have access to conventional sensor information including Radar, AIS, ECDIS and others.

The paper presents the methodology used to deduct, from the observations of fixations, when the navigator became aware of particular objects. It details how the human outlook is compared with that of the technology solution. On the technology side, the paper presents approaches to detection and classification of objects, which appeared to be efficient in coastal areas with confined passages and shallow water. The quality of outlook in different weather conditions is discussed and forms part of the comparison.

Keywords: Outlook for navigation, autonomous vessels, electronic outlook.

1 Introduction

Look-out for navigation is the task of observing various objects which can have an impact on a ships planned route and maneuvering capabilities, for example other

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **(mb, sh, adllo)@elektro.dtu.dk**

[2]AUT Group, Department of Applied Mathematics and Computer Science, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **stet@elektro.dtu.dk**

[3]Force Technology, **(tsk,jebs)@force.dk**

[4]Danish Maritime Authority, **cjb@dma.dk**

vessels, buoys and land. If the outlook is a seperate person on the bridge, observations are reported to the officer in charge who decide any remedial actions. The look-out is made using sight and aided by available technology such as radar, AIS and ECDIS systems. Development within camera technology and computer vision algorithms has provided an additional possible source for look-out. This paper investigates the quality of this "electronic outlook" and compares with human look-out.

2 Methods

2.1 Outlook for navigation

The analysis of manual lookout/watchkeeping is based on a combination of (1) on board observations, (2) recordings of the officer's visual attention using eye-tracking (Tobii Glasses Pro 2 wearable eye tracker and iMotions software for recordings) and (3) general maritime human factors knowledge.

The look-out task involves both endogenous- and exogenous-driven activities. Endogenous activities are visual attention controlled by the navigator himself on his own initiative and based on relevant knowledge and experience, such as observing navigational markings, sighting of land and watching out for other vessels. Exogenous activities are caused by an external event catching the attention of the navigator: the sight of a vessel which the navigator has not been looking for or by light or sound signals. Scenarios will typically be a combination of endogenous and exogenous look-out activities.

2.2 Eye tracking

Analysis of manual look-out is based on observations on board ferries in Danish costal regions during the summer 2018 and on general observations on board a large number of vessels during the period 2000-2018. Observations from ship simulator exercises during the same period and general knowledge of maritime human factors, is also included. For maritime use, eye-tracking has been used for simulator assessment of human reactions [1], [2], [3] and [4]. In [5] the focus is on evaluating and enhancing the training of navigators, but also enhancement of situation awareness has also been investigated using eye-tracking [6]. Using eye-tracking on-board vessels in traffic has not been reported earlier.

2.3 Electronic outlook

The electronic outlook system in this comparison consist of 5 cameras, a FMCW radar and an AIS receiver for reference. The vision system is composed of 2 color cameras (JAI GO-5000C), 2 monochrome cameras (JAI GO-5000M) with longpass filters for the NIR range and 1 IR camera (Teledyne Dalsa Calibir 640). The equipment is mounted on a forward facing stand on board the ferries. Object detection and classification algorithms are run as post-processing of the images.

2.4 Object detection and classification

Object detection and tracking in a maritime environment seems to be a well-explored area, and several previous works addresses this. Challenges includes waves that can cause a rapid change in the frame of reference [7], sudden change of illumination and unwanted reflections from the water [8], and the possibility of poor weather conditions that reduces the range of sight. As mentioned in the survey papers [9], [10] there are a range of methods that deals with detection and classification in images, and horizon line detection and background subtraction seems be effective for object detection [11], [12]. Methods are proposing to utilize infrared and visible light images [9], but also thermal imaging seems to have the ability to provide information about objects on the water [13]. With recent progress in deep learning based segmentation and classification methods, visible light images is an obvious choice for object detection since much training data, such as e.g. ImageNet [14], already exists and can provide a good base for training. For specifically maritime environments [15] and [16] shows that deep learning based methods are effective. In our project, we use the Mask-RCNN [17] on visible light images, which is able to perform pixel-wise segmentation of several classes. We argue that additional sensors such as near infrared and thermal imaging can provide additional valuable information, but has not yet been included in the classification pipeline in this stage of the project.

2.5 Mask-RCNN detection and classification

Objects that are within visual range of the cameras are detected and classified using a Convolutional Neural Network (CNN), also referred to as deep learning technology. The network architecture employed in this project to detect different objects in a maritime environment is Mask-RCNN [17], which has the novelty of not only being able to recognize and detect (bounding box) several classes, but is also able to segment all instances of each one and create the corresponding binary mask at a pixel level, in real time. Mask-RCNN is the culmination of an architectural model that started with a Region-Based Convolutional Neural Network (RCNN) [18], followed by Fast-RCNN [19] and then Faster-RCNN [20]. The Mask-RCNN architecture, was employed in this study, which is shown in Figure P6.1. It has branch at the end of the CNN that predicts mask-segmentation at a pixel-to-pixel level. Misalignment and loss of data is avoided, and pixel level precision is obtained by a RoI-Align layer that uses bilinear interpolation to remove quantization at the region of interest boundaries.

2.6 Training

In a previous project, Mask-RCNN was trained on the SUN-RGBD dataset but it was observed that adding a Depth input to the network was not significantly beneficial; thus, depth was excluded from the training of this project. Mask-RCNN has been pretrained using the weights from the COCO datatset [21]. Initially, the network was fine-tuned for the detection of 5 maritime classes: sea, land, sky, boat and buoy. Up to 200 images were hand-labelled using the LabelMe framework. The network was trained for 10 epochs on the first 4 layers (classificatory), then another 10 epochs for the rest of the layers and finally 20 epochs for the whole network. The learning rate was set to 0.001 and the momentum was 0.9. Training took 6 hours on a GTX GeForce 1080 GPU. Upon

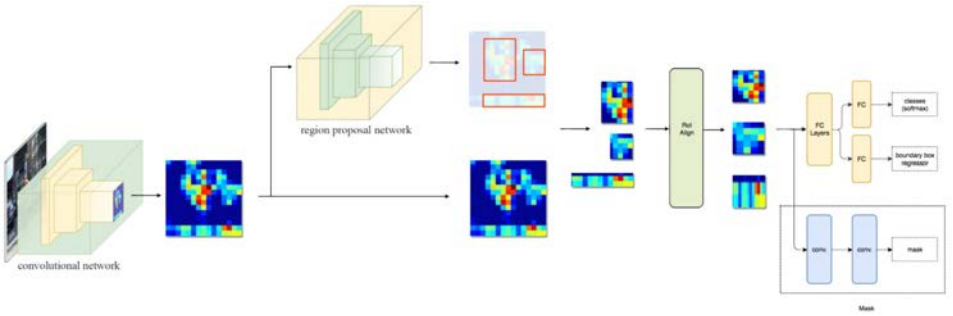


Figure P6.1: Mask-RCNN network. (*From: https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272*)

inspection, it was seen that the network had overfit in great measure. The reason for this was the limited amount of object instances in the training data compared to the training epochs carried out. In the following experiment, the number of classes was reduced to 2 (buoy and ship) and the number of labels was increased to about 600 images.

3 Results

This study compares the human outlook by assessing the fixations determined by the eye-tracking system with object classifications made by the electronic outlook.

Comparison between the electronic systems outlook capabilities and the human counterpart are hence done looking at the instant of first observations of a given object. The eye-tracking software gives an indication of fixation on an object when the human lookout has been gazing at it for a certain length of time. This time is compared to the timestamp that the Mask-RCNN indicates its first detection and classification of the object. Figure P6.2 shows a snapshot of of eye-tracking. The right part shows what the lookout is focusing on. The yellow line on this shows that the eyes wanders around, which is normal. Fixation is indicated by the red circle. The Electronic Outlook is illustrated in Figure P6.3.

Figure P6.2: Eye-tracking of the manual look-outs fixations. Left: Forward facing camera used as reference in the analysis. Right: Eye-tracking result. The yellow spot surrounded by a thin red line indicates fixation on an object.

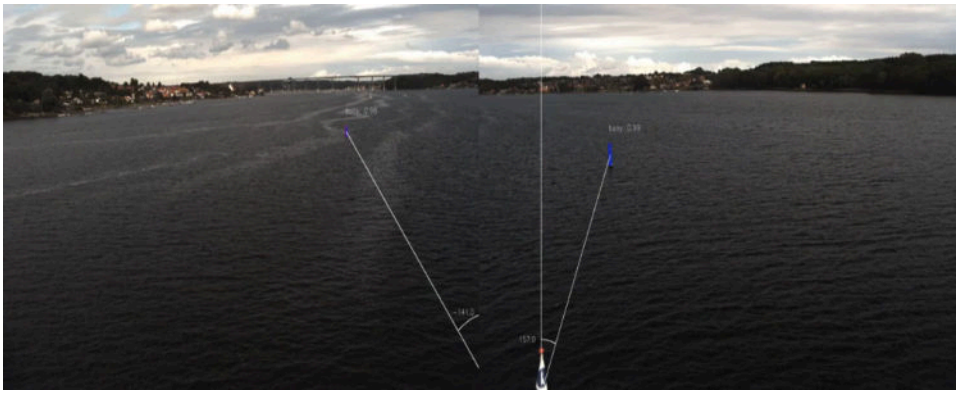


Figure P6.3: Object detection and classification on two RGB images are shown by highlighting the detected object in green colour and showing the bearing to detected objects.

The full paper will show detailed statistics and histograms for the comparison of object detection by the navigator and by the electro-optical sensor system. The full paper will also detail on limitations on some of the cameras in conditions of sparse daylight, and will discuss instrumentation needed in different weather and light conditions.

4 Discussion

Since the ship has a radar and AIS sensors on board, the detection of objects that are visible to RADAR or have AIS transmitters installed, could be done quite accurately. However, several objects are not visible on RADAR, such as leisure surf borders and sea kayaks, boats without RADAR reflector and AIS transmitter, and even containers that were accidentally dropped over board. Electronic outlook with object classification is therefore very important for the ship so that it acts in a safe manner also when non RADAR detectable objects are in the area. Thus, a combination of object positions from these sensors and the Mask-RCNN architecture could increase the performance and the results. An example of this is by using the detected objects positions from the radar as possible region proposals in the network.

Further results will therefore fuse onboard radar and AIS information to improve the performance of the vision system. This will require calibration that enables radar and AIS data to be projected from their respective coordinate systems into e.g. the pixel-coordinates of the input images to the CNN. This data could be used for region proposal in the network and be particularly useful in situations with reduced visibility of the cameras.

4.1 Coverage of this analysis

Some kinds of behaviour are related to look-out, which are not captured by only observing the areas of fixtures with eye tracking glasses, but require further interpretation:

- General visual observation (watching) of nothing in particular, but often focused on the direction of the vessel and abeam/passed in relation to the progression of the navigation.
- Exogenous-oriented attention in relation to above item 1 – something turns up. This can include comparison or verification with information from instruments e.g. radar or AIS.
- Endogenous-driven observation of objects from other sources – for instance sea charts (buoys), radar or AIS (vessels) – expected to be observable.

Such interpretation of the situation, which is part of a situational awareness scenario, is not yet part of this study.

Repeated observations to determine if there is a risk of collision and in this connection take countermeasures is visible in the eye-tracking measurements, and so is repeated observations to determine if countermeasures have the desired effect, but such awareness behaviours of the navigator are also outside the scope of the object detection and classification presented in this paper.

4.2 Electronic outlook as a fifth sense

Look-out is just one among several tasks of the navigator on the bridge. Other tasks include: Observation of the condition of engines and systems; Handling of cargo and passengers; Safety-related routines; Communication internally on board the vessel and with external parties; Management of staff and other administrative tasks; QA and documentation tasks; Handling of safety-critical situations on board.

With several other tasks to care for, which might sometimes distract the navigator, it is believed that electronic outlook could serve as a *fifth sense* for the navigator and perhaps make it possible to have temporally unmanned bridge in conditions with little to no other traffic.

5 Conclusions

This study compared human outlook with electronic. Using instance of fixation of eye-tracking glasses with instance of electronic outlook by cameras and mask-RCNN classification, the study provided statistics for a comparison on one of the essential parameters. The situational awareness elements in a comparison were not covered but will be the subject of further research.

Acknowledgements

The authors would like to acknowledge the dedicated efforts made by laboratory engineers, present and former students. This research was initiated by the Danish Maritime Authority and funded by the Danish Maritime Foundation via DTU's Maritime Centre. This funding is gratefully appreciated.

Bibliography

[1] F. Bjørneseth, C. Loraine, M. Dunlop, and S. Komandur, "Towards an understanding of operator focus using eye-tracking in safety-critical maritime settings," in *Proc. Int. Conference on Human Factors in Ship Design & Operation*, Glasgow, February 2014.

[2] S. Aleem, "Analysis of shiphandlers' eye-gaze and simulation data for improvements in cove-its system," Ph.D. dissertation, Naval Postgraduate School,, Monterey, California, 2017.

[3] O. Hareide and R. Ostnes, "Scan pattern for the maritime navigator," *Transnav – the International Journal on Marine Navigation and Safety of Sea Transportation*, 2017.

[4] S. Renganayagalu and S. Komandur, "Video support tools for training in maritime simulators," in *Proc. of the International Conference on Contemporary Ergonomics and Human Factors*, Cambridge UK, April 2013.

[5] B. Muczynski, M. Gucma, M. Bilewski, and P. Zalewski, "Using eye tracking data for evaluation and improvement of training process on ship's navigational bridge simulator," *Maritime University of Szczecin*, vol. 33, no. 105, pp. 75–78, 2013.

[6] M. Pico, H. D., R. Bik, S. van der Wiel, and R. van Basten Batenburg, "Enhancing situational awareness. a research about improvement of situational awareness on board of vessels." . Rotterdam Mainport University of Applied Sciences, Tech. Rep., 2015.

[7] S. Fefilatyev, D. Goldgof, M. Shreve, and C. Lembke, "Detection and tracking of ships in open sea with rapidly moving buoy-mounted camera system," *Ocean Engineering*, vol. 54, pp. 1–12, 2012.

[8] D. D. Bloisi, A. Pennisi, and L. Iocchi, "Background modeling in the maritime domain," *Machine vision and applications*, vol. 25, no. 5, pp. 1257–1269, 2014.

[9] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: a survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 1993–2016, 2017.

[10] R. D. S. Moreira, N. F. F. Ebecken, A. S. Alves, F. Livernet, and A. Campillo-Navetti, "A survey on video detection and tracking of maritime vessels," *International Journal of Recent Research and Applied Studies*, vol. 20, no. 1, 2014.

[11] Y. Zhang, Q.-Z. Li, and F.-N. Zang, "Ship detection for visual maritime surveillance from non-stationary platforms," *Ocean Engineering*, vol. 141, pp. 53–63, 2017.

[12] Z. L. Szpak and J. R. Tapamo, "Maritime surveillance: Tracking ships inside a dynamic background using a fast level-set," *Expert systems with applications*, vol. 38, no. 6, pp. 6669–6680, 2011.

[13] F. S. Leira, T. A. Johansen, and T. I. Fossen, "Automatic detection, classification and tracking of objects in the ocean surface from uavs using a thermal camera," in *Aerospace Conference, 2015 IEEE*. IEEE, 2015, pp. 1–10.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255.

[15] M. Leclerc, R. Tharmarasa, M. C. Florea, A.-C. Boury-Brisset, T. Kirubarajan, and N. Duclos-Hindié, "Ship classification using deep learning techniques for maritime target tracking," in *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, 2018, pp. 737–744.

[16] F. Bousetouane and B. Morris, "Fast cnn surveillance pipeline for fine-grained vessel classification and detection in maritime scenarios," in *Advanced Video and Signal Based Surveillance (AVSS), 2016 13th IEEE International Conference on*. IEEE, 2016, pp. 242–248.

[17] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[19] R. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, p. 1137–1149, Jan 2017.

[21] 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*. Springer, 2014, pp. 740–755.

[22] S. P. van den Broek, H. Bouma, R. J. den Hollander, H. E. Veerman, K. W. Benoist, and P. B. Schwering, "Ship recognition for improved persistent tracking with descriptor localization and compact representations," in *Electro-Optical and Infrared Systems: Technology and Applications XI*, vol. 9249. International Society for Optics and Photonics, 2014, p. 92490N.

[23] L. P. Perera, P. Oliveira, and C. G. Soares, "Maritime traffic monitoring based on vessel detection, tracking, state estimation, and trajectory prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1188–1200, 2012.

[24] T. Porathe, J. Prison, and Y. Man, "Situation awareness in remote control centres for unmanned ships," in *Proceedings of Human Factors in Ship Design & Operation, 26-27 February 2014, London, UK*, 2014, p. 93.

[25] J. Tang, C. Deng, G.-B. Huang, and B. Zhao, "Compressed-domain ship detection on spaceborne optical image using deep neural network and extreme learning machine," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 3, pp. 1174–1185, 2015.

[26] E. Tu, G. Zhang, L. Rachmawati, E. Rajabally, and G.-B. Huang, "Exploiting ais data for intelligent maritime navigation: A comprehensive survey," *arXiv preprint arXiv:1606.00981*, 2016.

Appendix A1

MatLab Simulation of a Rao-Blackwellized Particle Filter with Improved Techniques for Grid Mapping in Mobile robots

Adrian Llopart[1]

Abstract

Rao-Blackwellized Particle filters have been introduced in simultaneous localization and mapping (SLAM) effectively in the past years. Improved techniques have helped reduce the number of particles needed and allowed a faster and more robust solution. These solutions can be seen, and used, in the C++ ROS *gmapping* package. This paper presents the conversion to a simpler but easy-to-understand MatLab implementation and will describe the RBPF-SLAM principle in a similar manner.

1 Introduction

One of the clearest functionalities of mobile robots is building maps of its surroundings and navigating through them; this is often referred to as SLAM, Simultaneous Localization and Mapping. There exists, however, certain difficulties when implementing SLAM, mainly, for a robot to localize (know its true position), a very precise map has to be built before; but, for a very precise map to be built, a robot must know exactly where it is. Friction, control loss or small obstacles are often the cause of a bad odometry which leads to a poor estimate of the true position. The Rao-Blackwellized Particle filter (as shown in [1] and [2]) solves this issue by generating estimates of the possible position (particles) and giving them a weight. Those particles that have a higher weight, because their estimate matches better the reality, will survive, and the rest will die out in the next generation. To keep a continuous amount of particles and not let all of them die out, resampling stages are carried out where those surviving particles reproduce and keep the particle count constant. It is evident that the more particles used, the more possible descriptions of the reality one has and the higher probability of having at

[1]AUT Group, Department of Electrical Engineering, DTU, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark, **adllo@elektro.dtu.dk**

least one particle which is almost perfect. This also induces a higher computational necessity and stops the solution from becoming a real-time application.

The real issue is, therefore, weighing these particles to know how close they are to the reality. This is known as the *proposal distribution* π, and in the latter iterations of the RBPF-SLAM solution, optimal techniques have been developed to obtain a more accurate representation, whilst keeping the particle count to a minimum.

In summary, what the RBPF-SLAM algorithm does is given an initial estimate of the true pose, it will use the high precision of a laser scan and the latest map generation to converge the estimate to the true position. This leads to a more accurate map representation and a considerable reduction in errors and number of particles.

2 RBPF joint posterior

As presented by [2], the key idea of RBPF-SLAM is to estimate the joint posterior $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$. In other words, the algorithm should compute the true (or as close as possible) position (x) of the robot and a map (m) for every iteration (on a per particle basis), given solely the laser scan data (z) and the odometry measurements (u). This posterior can be factorized as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{A1.1}$$

This allows us to estimate the position of the robot using the logged odometry and the laser scan data, and then use this position and the laser data again to generate more parts of the map. Thus, the generated map depends heavily on the estimate of the position. This technique is known as Rao-Blackwellization.

2.1 Odometry distribution

The odometry distribution used in this paper implements the approach *Velocity Motion Model* presented in [3] (pages 121-132), where the control is defined as:

$$u_t = \{v_t w_t\}$$

where v_t defines the *translational velocity* and w_t the *rotational velocity*. Deriving the probability of one specific position x_t being fulfilled after one specific initial position x_{t-1} and control command u_t, and sampling from said motion model, are depicted in algorithm **motion_model_velocity(x_t,u_t,x_{t-1})** (page 123) and algorithm **sample_motion_model_velocity(u_t,x_{t-1})** (page 124), respectively.

2.2 Posterior over maps

The posterior over maps $p(m|x_{1:t}, z_{1:t})$ can be computed analytically using several methods. This paper used the approach presented by [3] known as *Occupancy grid mapping* (Algorithm 7) and the *Inverse sensor model* (pages 284-292) to generate a 2-D floor plan occupancy grid map.

Therefore, a map m is partitioned into finitely many grid cells m_i, where each cell contains a value that refers to the probability of it being occupied $p(m_i)$ with values ranging from '0' (free) to '1' (fully occupied). The posterior over maps can then be

approximated as the product of its marginals.

$$p(m|x_{1:t}, z_{1:t}) = \prod_i p(m_i|x_{1:t}, z_{1:t})$$

It is worth noting that this algorithm will use the *log-odds* representation of occupancy to avoid numerical instabilities for probabilities close to zero or one:

$$l_{t,i} = log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}$$

and that the probability ration can be easily recovered from the *log-odds* ratio as:

$$p(m_i|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + exp(l_{t,i})}$$

Finally, the algorithm makes use of the variables l_0, $l_{occupied}$ and l_{free} which can be adjusted as one pleases. The values applied for this simulation are 0.5, 1 and -1. Also, α makes reference to the average width of walls; and β is the width of the sensor beam. Considering that the simulation uses one cell to represent $10cm^2$, α is set to be equal to 2 (thus 20 cm^2) and β will be equal to 0.1.

Algorithm 7 Complete Occupancy Grid Mapping

procedure OCCUPANCY_GRID_MAPPING ($l_{t-1,i}$, x_t, z_t)
  for all cells m_i **do**
    if m_i in perceptual field of z_t **then**
      $l_{t,i} = l_{t-1,i}$ + **inverse_range_sensor_model(**m_i, x_t, z_t**)** $- l_0$
    else
      $l_{t,i} = l_{t-1,i}$
    end if
  end forreturn $l_{t,i}$
end procedure

procedure INVERSE_RANGE_SENSOR_MODEL (m_i, x_t, z_t)
  Let x_i, y_ibe the center of mass of m_i
  $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
  $\phi = atan2(y_i - y, x_i - x) - \theta$
  $k = argmin_j |\phi - \theta_{j,sens}|$
  if $r > min(z_{max}, z_t^k + \alpha/2) or |\phi - \theta_{j,sens} > \beta/2|$ **then return** l_0
  end if
  if $z_t^k > z_{max} and |r - z_t^k| < \alpha/2$ **then return** $l_{occupied}$
  end if
  if $r \leq z_t^k$ **then return** l_{free}
  end if
end procedure

2.3 Posterior over potential trajectories

The posterior over potential trajectories $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ will be solved applying a particle filter. This means that one particle will represent one potential trajectory over one time step while generating its own map.

Usually, a non-optimal particle filter called SIR (sampling importance resampling) is applied. This filter follows 4 main steps:

- *Sampling*: The next generation of particles $x_t^{(i)}$ is obtained from the previous generation $x_{t-1}^{(i)}$ by sampling from the proposal distribution π. Normally, a probabilistic odometry motion model is used here. This paper will later suggest improved techniques for achieving an optimal proposal distribution.

- *Importance weighing*: Each particle is assigned with a weight $w_t^{(i)}$ according to the importance sampling principle:

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} \tag{A1.2}$$

  which incorporates all observations and positions up until that specific point in time.

- *Resampling*: New particles are drawn which replace the old ones depending on their weight. This keeps the particle count constant. The new generation of particles have all the same initial weight.

- *Map estimation*: For every particle, a map is generated according to the approach presented before (Algorithm 7).

One might rapidly notice that this filter requires a trajectory weight evaluation after every observation/time step. Hence, this schema becomes highly inefficient over time as trajectories grow. [4] obtained a recursive formulation to compute the importance weights, which allows the calculation of the next weight based on the previous one (and not having to calculate over and over again all weights until that point):

$$w_t^{(i)} = \frac{p(z_t|m_{t-1}^{(i)}, x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1})}{\pi(x_t|x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1})} \cdot w_{t-1}^{(i)} \tag{A1.3}$$

Generic particle particle filters rely heavily on the recursive structure of equation A1.3, whilst leaving open what the proposal distribution will be and when the resampling should take place. This paper will focus on the improved techniques presented in [5] to obtain an optimal proposal distribution based on both odometry and sensor data.

3 RBPF Improved proposal

As described previously, one needs to draw particles from the proposal distribution π to obtain an estimate on the next generation of particles. Logically, the more accurate the proposal is, the better the end result of the estimation. Typically, particle filters use

the odometry proposal distribution because it is easily computed and replacing said odometry distribution in equation A1.3 yields a very simple weight calculation:

$$w_t^{(i)} = p(z_t|m_{t-1}^{(i)}, x_t^{(i)}) \cdot w_{t-1}^{(i)} \tag{A1.4}$$

However, this approach leads to a suboptimal result. Specifically because the precision of the laser scan data is not taken into account. In other words, if we only sample from the odometry proposal, the importance weights between particles will end up differing significantly because the drawn samples only cover a fraction of the state space region, as shown in Figure A1.1. If the laser scan data was incorporated, and knowing that it provides a very pinpointed area of high likelihood, the resulting estimation would be significantly more precise. Therefore, integrating the sensor data z_t into the proposal

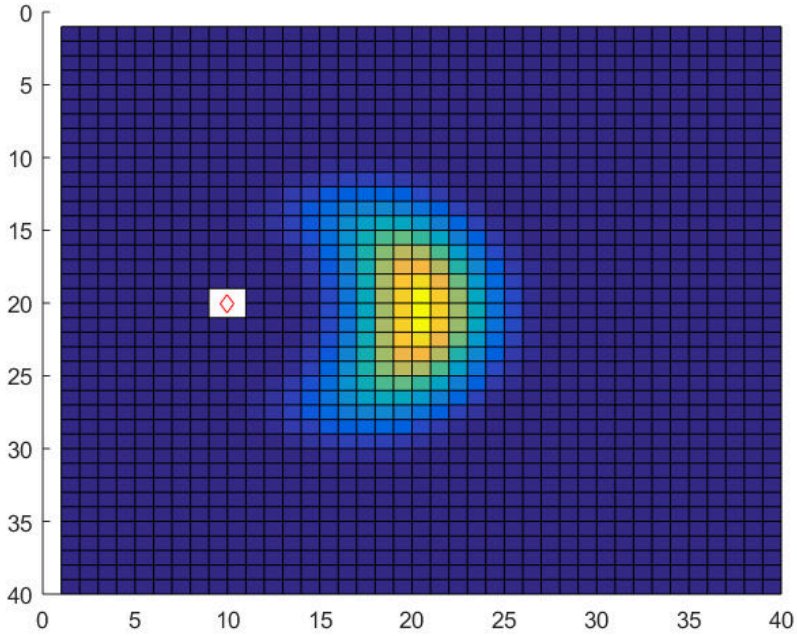


Figure A1.1: Odometry proposal distribution for a command $u_t = [100]$. The colors represent the probability of the robot being in that position after running the specified command

will focus the sampling only in the meaningful regions of the observation likelihood. The resulting distribution, according to [6] is as follows:

$$p(x_t|x_{t-1}^{(i)}, m^{(i)}, z_t, u_t) = \frac{p(z_t|x_t, m^{(i)})p(x_t|x_{t-1}^{(i)}, u_t)}{p(z_t|x_{t-1}^{(i)}, m^{(i)}, u_t)} \tag{A1.5}$$

It is common practice to redefine the target distribution in equation A1.5 as:

$$\tau(x_t) = p(z_t|x_t, m^{(i)})p(x_t|x_{t-1}^{(i)}, u_t)$$

The major problem this approach has with grid maps is that the closed-form approximation of the proposal is not available due to an unpredictable shape of the observation likelihood function.
This could be solved applying an *adaptive* particle filter which samples potential poses x_j from the motion model and then weights them through the observation likelihood to obtain an approximation of the optimal proposal. Since the observation likelihood is typically peaked in very small areas, a dense sampling is necessary to capture correctly the peak, leading to a sub-optimal proposal, much like the motion model, because of its high computational requirements. Thus this paper will not pursue this approach.

Another way of solving this major issue is by taking into consideration that the observation likelihood typically has only one maxima allowing us to sample around this maxima and avoiding other less meaningful regions. Specifically, the posterior $p(x_t|x_{t-1}^{(i)}, m^{(i)}, z_t, u_t)$ is locally approximated around the maxima of the observation likelihood via a *scan matching* procedure.

3.1 Scan matching approach

Initially, the next generation of particles is sampled from the odometry distribution. These particles are then approximated to the meaningful area of the observation likelihood via a *Scan matching* method. In this process, the current scan data z_t is compared to the particles map generated up until this point m_i to converge the initial guess of the particles position to one that is closer to the reality. The algorithm used in this paper is known as ICP (Iterative Closest Point) which returns translational and rotational vectors that are applied to x_t to reach a better position estimate. The algorithm takes into account two sets of point clouds and iteratively overlays them by minimizing the squares error.

It is necessary to sample around this new estimated position. To do so, first a Gaussian has to be determined around it. To calculate the mean $\mu^{(i)}$ and the variance $\Sigma^{(i)}$ we also take into account the odometry information:

$$\mu^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \tag{A1.6}$$

$$\Sigma_t^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \cdot (x_j - \mu_t^{(i)})(x_j - \mu_t^{(i)})^T \tag{A1.7}$$

with the normalization factor being:

$$\eta^{(i)} = \sum_{j=1}^{K} p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) = \sum_{j=1}^{K} \tau(x_t) \tag{A1.8}$$

Now we have a closed-form approximation of the optimal proposal. As shown in [5], using the new proposal distribution, the new weights are calculated as:

$$w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)} = w_{t-1}^{(i)} \cdot \sum_{j=1}^{K} \tau(x_t) \tag{A1.9}$$

The main advantage of the improved proposal is that it takes both the odometry data and laser scans into consideration which reduces notably the proposals densities uncertainty and allows more efficient sampling. The only problem this approach presents is when the observation likelihood is multimodal. Since the scan matching algorithm maximizes the observation likelihood around the closest local maxima to the initial guess, it is possible that this approximation fails because it misses other additional maxima due to the multimodlity of the likelihood. In other words, the reported local maxima is not the global maxima. However, in reality, the distribution tends to be uni-modal, as reported in [5], and so this issue is of little concern in common practices.

Another problem is when the scan matcher fails to report a scan conversion. This is generally because of poor observations or low overlapping areas between scans. For this case in particular, the proposal will only be determined by the raw motion model. This issue, however, is, in reality, not very common either.

4 RBPF Adaptive resampling

The second main difference, this paper presents, with other general particle filters concerns the resampling step. During resampling, particles with low weight are replaced by new ones with higher weight. This is essential because only a finite amount of particles are used to approximate the target distribution; but, if not applied carefully, the resampling can remove good samples and lead to impoverishment of the particle filter. Also, the criterion for deciding when to resample is of utmost importance for a good overall performance.

The approach used in this paper relies in the so-called effective sampling size for target posterior estimation [7]. This value was proposed by [4] to be:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\bar{w}^{(i)})^2} \tag{A1.10}$$

where $\bar{w}^{(i)}$ is the normalized weight of the particle i.

The key idea behind N_{eff} is that the worse the particle approximation is, the larger variance between weights there would be. If the estimate was perfect, all particles would end up with the same weight value. As presented in [5], the resampling step should carried out when the value of N_{eff} drops below $N/2$, where N is the total number of particles.

5 The RBPF improved algorithm

The algorithm 8 presents the full schema for the improved RBPF utilized in this paper. The steps are as follows:

- An initial estimate of each particles position $x_t^{(i)}$ is obtained from the previous pose $x_{t-1}^{(i)}$, the odometry measurements u_{t-1} and a realistic approximation of the errors the odometry might propose.
- A scan matching algorithm, in this case an ICP, is executed between the latest laser scan data z_t and a point cloud derived form the latest map generated for that particle. To calculate the point cloud from the map, only that area inside

what would be the region of the laser scanner is evaluated. This saves important resources and does not calculate point clouds in regions that cannot be matched with the laser scan data. The rotational and translational vectors derived through the ICP are used to reconstruct a new pose estimate $\dot{x}_t^{(i)}$ from the motion model estimate $x_t^{(i)}$. Finally, if the scan matcher fails at converging both point clouds, the pose estimate will remain as the one obtained from the motion model.

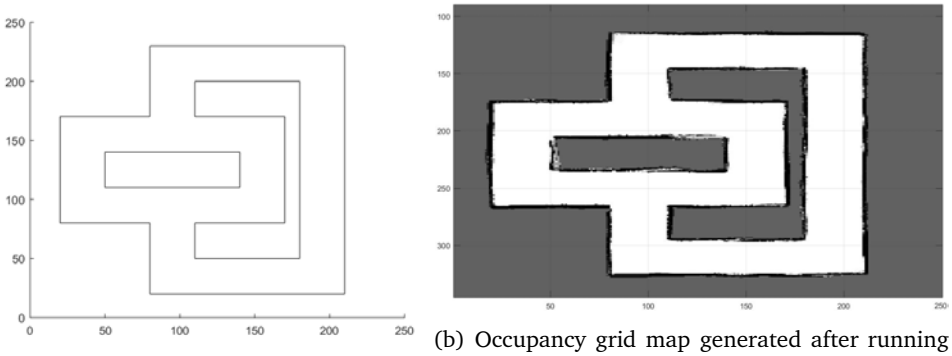
- A set of sampling points x_j is selected in an interval around $\dot{x}_t^{(i)}$. The mean and covariance values for creating a Gaussian distribution are calculated as in equations A1.6 and A1.7 via a pointwise evaluation of the target proposal. The first half of the target proposal, $p(x_j|x_{t-1}^{(i)}, u_t)$, is obtained by evaluating the sample x_j in the odometry motion model distribution. The second half of the target distribution, $p(z_t|x_j, m^{(i)})$, is obtained evaluating the observation likelihood for that sample. Several approaches can be taken; this paper has used a point to point comparison between the true scan and the estimated scan a robot in the estimated position in the generated map would see. In this stage the weighing factor $\eta^{(i)}$ is also computed.

- A new final pose estimate $\ddot{x}_t^{(i)}$ for each particle is drawn from the Gaussian approximation $\mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$ of the improved proposal.

- The importance weights are calculated as in equation A1.9.

- The particles map m_i is updated with the final estimated position $\ddot{x}_t^{(i)}$ and the laser scan data z_t.

- The resampling stage is carried out depending on the value of N_{eff}.



(a) Map created by the user

(b) Occupancy grid map generated after running RBPF-SLAM

Figure A1.2: Input and output maps for the Rao-Blackwellized Particle Filter Grid SLAM

Algorithm 8 Improved techniques for RBPF in grid mapping

procedure RBPF-SLAM (x_{t-1}, z_t, u_t, m_{t-1})
  $\mathcal{S}_{t-1}$: sample set of the previous time step
  for $\mathcal{S}_{t-1}^{(i)} \in \mathcal{S}_{t-1}$ **do**
    $< x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} >= \mathcal{S}_{t-1}^{(i)}$
    // initial pose estimate
    $x_t^{(i)} = x_{t-1}^{(i)} \oplus u_t$
    // scan matching
    $\dot{x}_t^{(i)} = \text{argmax}_x p(x|m_{t-1}^{(i)}, z_t.x_t^{(i)})$
    if $dotx_t^{(i)} =$ failure **then**
      $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$
      $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t|x_t, m^{(i)})$
    else
      // sample around node
      for $k = 1, ..., K$ **do**
        $x_k \sim \left\{ x_j || x_j - \dot{x}_t^{(i)}| < \Delta \right\}$
      end for
      // compute gaussian
      for all $x_j \in \{x_1, ..., x_K\}$ **do**
        $\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1})$
        $\eta^{(i)} = \eta^{(i)} + p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1})$
      end for
      $\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
      $\Sigma_t^{(i)} = 0$
      for all $x_j \in \{x_1, ..., x_K\}$ **do**
        $\Sigma_t^{(i)} = \Sigma_t^{(i)} + p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \cdot (x_j - \mu_t^{(i)})(x_j - \mu_t^{(i)})^T$
      end for
      $\Sigma_t^{(i)} = \Sigma_t^{(i)} / \eta * (i)$
      // sample new pose from gaussian
      $x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$
      // update importance weights
      $w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
    end if
    // update map
    $m_t^{(i)} = \text{Occupancy\_Grid\_Mapping}(m_{t-1}^{(i)}, x_t, z_t)$
    // update sample set
    $\mathcal{S}_t = \mathcal{S}_t \cup \left\{ < x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} > \right\}$
  end for
  $N_{eff} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
  if $N_{eff} < N/2$ **then**
    $\mathcal{S}_t =< textresample\mathcal{S}_t$
  end if
end procedure

6 Simulations and results

The *Matlab* code to which this paper refers to works with a simulated RBPF-SLAM. The user is able to create its own map (Figure A1.2a), generate a path which the robot should follow, adjust a series of parameters related to the motion model of the robot or related to the particle filter itself, and simulate the results for a RBPF-SLAM (Figure A1.2b). The resulting grid map will have a resolution of $10\frac{cm^2}{cell}$.

The errors generated via this schema can be seen in Figure A1.3. They are minimal except for a particular point in which *theta* overshoots. This is definitely due to misalignment's in *theta* for the case $0^o = 360^o$.

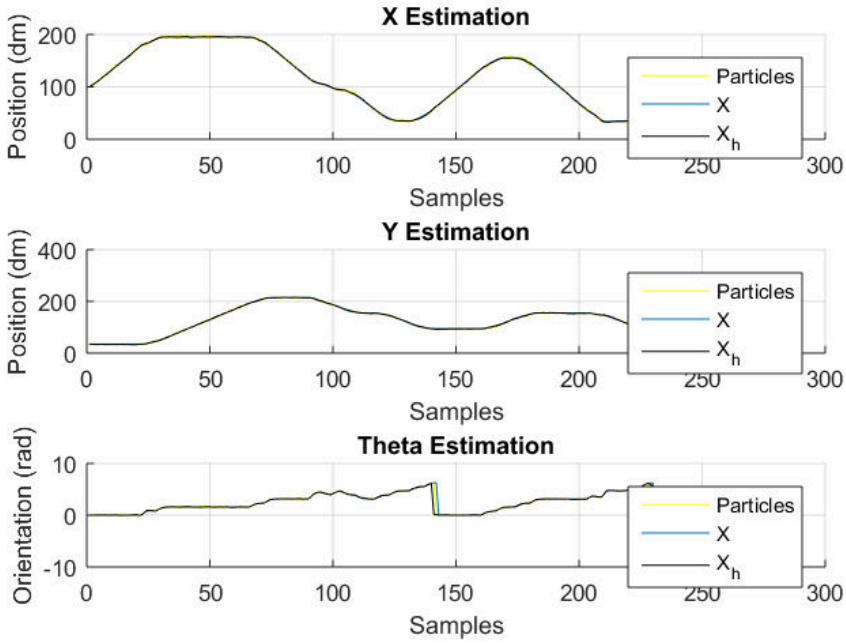


Figure A1.3: Error between the real and estimated x, y and *theta* values for one particle

7 Conclusions

The simulation results represent a functional and efficient RBPF-SLAM. Error position estimation error for each particle is minimal, and the misalignment's in the resulting map are negligible.

Some improvements can still be carried out: the resampling step occurs slightly too often due to using a suboptimal evaluation of $p(z_t|x_j, m^{(i)})$ leading to the motion model distribution affecting in a larger manner the proposal distribution, thus loosing partly the high precision of the laser scan data. Additionally, the implemented *scan*

matching technique could be improved. Paper [5] presents the scan-matcher *'vasco'*, which is part of the Carnegie Mellon Robot Navigation Toolkit (CARMEN) [8].

Finally, the *MatLab* code implemented has been categorized as user-friendly because of its simplicity in use and the detailed User Manual attached.

Bibliography

[1] A. Doucet, J. de Freitas, K. Murphy, and S. Rusel, "Rao-blackwellized particle filtering for dynamic bayesian networks," in *Conf. Uncertainty Artif. Intell. Stanford, CA*, 2000, pp. 173–186.

[2] K. Murphy, "Bayesian map learning in dynamic environmnets," in *Conf. Neural Inf. Process. Syst. Denver, CO*, 1999, pp. 1015–1021.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006.

[4] A. Doucet, N. de Freitas, and N. Gordan, *Sequential Monte-Carlo Methods in Practice*. Springer-verlag, 2001.

[5] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," in *IEEE Trans. Robot.*, 2007.

[6] A. Doucet, "On sequential simulation-based methods for bayesian filtering," *Univ. Cambridge, Dept. Eng., Signal process. Group, cambridge, UK*, 1998.

[7] J. liul, "Metropolized independant sampling with comparisons to rejection sampling and importance sampling," in *Statist. Comput., vol 6*, 1996, pp. 113–119.

[8] M. Montemerlo, N. Roy, D. Hahnel, S. Thrun, C. Stachniss, and J. Glover, *CARMEN - The Carnegie Mellon Robot Navigation Toolkit*. [Online] http://carmen.sourceforge.net, 2002.

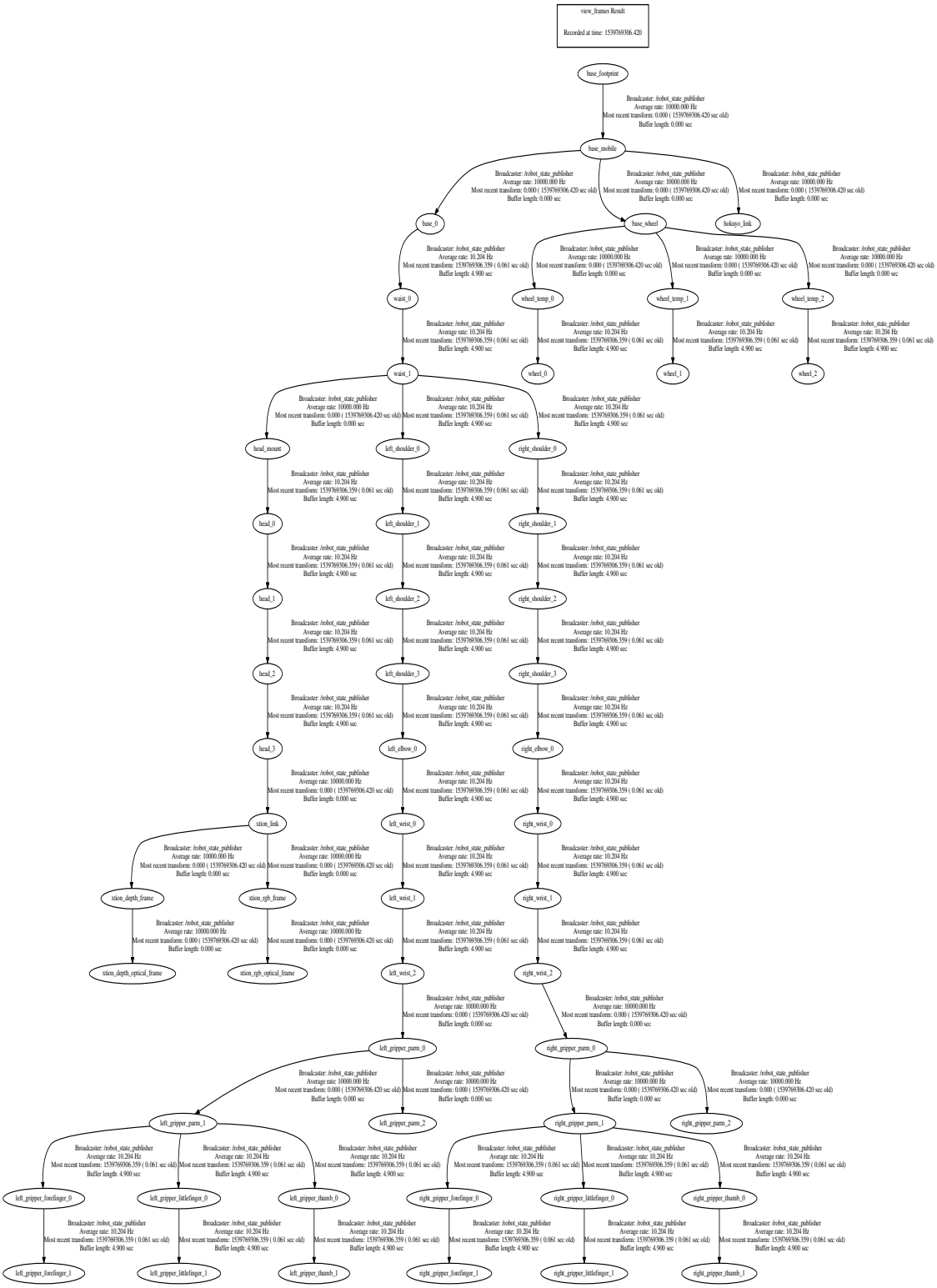
[9] A. Great, *This is the book title*. This is the name of the publisher, 2015.

[10] F. Author, S. Author, and P. Author, "This is the title of a conference paper," in *This is the title of the proceedings/conference*, 2015, pp. 1–100.

[11] M. Power, "This is the title of a journal article," *This is the name of the journal*, vol. 8, no. 1, pp. 2–3, 2015.

Appendix A2

MyBot humanoid robot relative poses between joints frames


view_frames Result
Recorded at time: 1539769306.420
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1539769306.420 sec old)
Buffer length: 0.000 sec
base_mobile
base_0
base_wheel
hokuyo_link
waist_0
wheel_temp_0
wheel_temp_1
wheel_temp_2
waist_1
wheel_0
wheel_1
wheel_2
head_mount
left_shoulder_0
right_shoulder_0
head_0
left_shoulder_1
right_shoulder_1
head_1
left_shoulder_2
right_shoulder_2
head_2
left_shoulder_3
right_shoulder_3
head_3
left_elbow_0
right_elbow_0
xtion_link
left_wrist_0
right_wrist_0
xtion_depth_frame
xtion_rgb_frame
left_wrist_1
right_wrist_1
xtion_depth_optical_frame
xtion_rgb_optical_frame
left_wrist_2
right_wrist_2
left_gripper_parm_0
right_gripper_parm_0
left_gripper_parm_1
left_gripper_parm_2
right_gripper_parm_1
right_gripper_parm_2
left_gripper_forefinger_0
left_gripper_littlefinger_0
left_gripper_thumb_0
right_gripper_forefinger_0
right_gripper_littlefinger_0
right_gripper_thumb_0
left_gripper_forefinger_1
left_gripper_littlefinger_1
left_gripper_thumb_1
right_gripper_forefinger_1
right_gripper_littlefinger_1
right_gripper_thumb_1

Technical University of Denmark
Department of Electrical Engineering
Section of Automation and Control
Elektrovej, Building 326
DK-2800 Kgs. Lyngby
Denmark
Phone: (+45) 45 25 34 76
Email: info@elektro.dtu.dk
www.elektro.dtu.dk