UNIVERSIDADE
BEIRA INTERIOR

Software for a Service Robot

André Rosa de Sousa Porfírio Correia

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Luís Filipe Barbosa de Almeida Alexandre

junho de 2020

Agradecimentos

Neste capítulo irei-me referir às pessoas que tiveram grande influência durante o desenvolvimento desta tese. Em primeiro lugar quero agradecer ao professor Luís Alexandre pelo enorme apoio, por todos os conselhos que me deu e por estar sempre disponível quando precisei, teria sido impossível sem a sua ajuda. Quero também agradecer à minha família que sempre me apoiou não só durante o desenvolvimento deste tese, como em toda a minha vida. Foram eles que me permitiram entrar e terminar o mestrado. Por último, queria agradecer a todos os meus amigos, em especial aos colegas do Soft Computing and Image Analysis Laboratory (SociaLab), nomeadamente ao Gaspar Ramôa, Vasco Lopes, António Gaspar, Bruno Silva e Nuno Pereira que me ajudaram durante o desenvolvimento desta tese.

Resumo

Os robôs de serviço são cada vez mais comuns devido aos avanços constantes na área da inteligência artificial, substituindo os humanos em tarefas cada vez mais complexas. Ao ter um robô autónomo e competente desempenhando tarefas diárias em vez do seu dono humano, a produtividade destes consequentemente aumenta. A pesquisa por melhores robôs de serviço levou à criação de competições robóticas onde tais tipos de robôs são avaliados e o estado da arte é forçado a avançar. Com o objetivo de possuir um robô de serviço que sirva as pessoas na universidade, bem como um dia participar em tais competições, o *Socialab* adquiriu um robô *Turtlebot2*. O laboratório tem proposto vários projetos ao longo dos anos, cada um adicionando novos níveis de funcionalidade ao robô. Com cada estudante que tem vindo a realizar o respetivo projeto, o software que foi desenvolvido tem estado continuamente a aumentar. Adicionalmente, cada projeto completado tem permanecido separado dos restantes e não tem sido utilizado desde o momento da sua criação. Por esta razão, o software desenvolvido está a ser desperdiçado. Portanto, é imperativo integrar todo o software disponível no robô. No entanto, como novos projetos serão desenvolvidos, este problema de projetos disjuntos poderá voltar a ocorrer após a integração dos projetos atuais. Ademais, com o aumento da funcionalidade que é desenvolvida, mais dificil e demorado será a sua integração. De forma a evitar este problema na sua totalidade, é necessário criar software estrutural que facilite o desenvolvimento de novas funcionalidades, bem como a sua integração com o software já existente.

De forma a atingir este objetivo, foi desenvolvida uma classe cujo propósito é controlar a execução de todos os processos em execução no robô. Adicionalmente, foi efetuada uma pesquisa sobre diversas competições robóticas com o objetivo de identificar os tipos de funcionalidades mais comuns, que referimos como módulos. Depois foi realizada uma implementação de cada um destes módulos. Devido à universalidade destes módulos, a sua implementação permite que software futuro, que provavelmente depende de alguns dos módulos, apenas os tenha que importar, em contraste com ter que os re-implementar. Adicionalmente, em linha com as práticas de qualidade de software, se cada um dos módulos precisa de uma atualização, esta apenas tem de ser realizada nos respetivos módulos, ao invés de ter de atualizar cada software adjacente que teve de o implementar. Este foi o primeiro objetivo da tese. Após a criação de uma fundação sólida para o desenvolvimento de software para robô, o foco transferiu-se para a criação de nova funcionalidade. Uma lista de tarefas robóticas foi obtida da pesquisa anterior sobre várias competições robóticas. A ideia é que se o robô é capaz de realizar tais tarefas então não só pode participar nas competições que as requerem, como também tem utilidade que pode ser utilizada pelas pessoas na universidade. A lista visava ser o mais extensa possível, tendo em conta as restrições temporais de desenvolvimento da tese, de modo a deixar o robô com o máximo de funcionalidades possível. Desta forma, sete tarefas foram escolhidas. A implementação de cada uma das tarefas é explicada em detalhe no seu capítulo respetivo.

À medida que cada tarefa foi desenvolvida, os seus componentes individuais foram extraídos para novos módulos, passiveis de serem utilizados por outras funcionalidades, respeitando assim, o objetivo inicial de criar software flexível e reutilizável. Este fator tornou a criação das tarefas seguintes cada vez mais simples devido a estas dependerem de funcionalidades já implementadas nas anteriores. A implementação das tarefas exigiu conhecimento das diferentes áreas da inteligência artificial. Este facto levou à ampliação do meu conhecimento ao invés da especialização numa área, algo que é frequente na realização de teses. A realização deste trabalho demonstra como as distintas tarefas robóticas foram implementadas. Devido à natureza variada das tarefas, é demonstrado como enfrentar um conjunto diverso de problemas que podem aparecer na área da inteligência artificial. Adicionalmente, este trabalho apresenta uma abordagem para a criação de uma fundação sólida para o desenvolvimento e integração de novo software. Por último, as tarefas estão aferidas contra o estado da arte, significando que atualizações futuras às tarefas podem ser realizadas e provadas superiores através da comparação dos seus resultados.

Palavras-chave

Robótica, Inteligência Artificial, Qualidade de Software, Segmentação Semântica, Classificação, Deteção

Resumo alargado

Os robôs de serviço são cada vez mais comuns devido aos avanços constantes na área da inteligência artificial, substituíndo os humanos em tarefas cada vez mais complexas. Ao ter um robô autónomo e competente desempenhando tarefas diárias em vez do humano, a produtividade destes consequentemente aumenta. A pesquisa por melhores robôs de serviço levou à criação de competições robóticas onde tais tipos de robôs são avaliados e o estado da arte avança. Com o objetivo de possuir um robô de serviço que sirva as pessoas na universidade, bem como um dia participar em tais competições, o *Socialab* adquiriu um robô *Turtlebot2*. O laboratório tem proposto vários projetos ao longo dos anos, cada um adicionando novos níveis de funcionalidade ao robô. Com cada estudante que tem vindo a realizar o respetivo projeto, o software que foi desenvolvido tem estado continuamente a aumentar. Adicionalmente, cada projeto completado tem permanecido separado dos restantes e não tem sido utilizado desde o momento da sua criação. Por esta razão, o software desenvolvido está a ser desperdiçado. Portanto, é imperativo integrar todo o software disponível no robô. No entanto, como novos projetos serão desenvolvidos, este problema de projetos disjuntos poderá voltar a ocorrer após a integração dos projetos atuais. Ademais, com o aumento da funcionalidade que é desenvolvida, mais dificil e demorado será a sua integração. De forma a evitar este problema na sua totalidade, é necessário criar software estrutural que facilite o desenvolvimento de novas funcionalidades, bem como a sua integração com o software já existente. Após a criação de uma fundação sólida para o desenvolvimento de software para robô, o foco transferiu-se para a criação de nova funcionalidade.

O primeiro capítulo define o âmbito e o problema onde esta dissertação se enquadra. Adicionalmente, são descritos os principais objetivos da realização deste trabalho, bem como as principais contribuições deste para a realização e integração de novos trabalhos cujo intuito seja o desenvolvimento de funcionalidades para o robô. Por fim, é explicada a estrutura deste documento.

O segundo capítulo foca a criação de uma fundação sólida para o desenvolvimento e integração de novo software. Nele é explicado o estado atual do software do robô e como foi atualizado. De seguida o funcionamento do ROS, ferramenta que facilita a criação de software para robôs, é explicado de forma sucinta. Após a explicação, é concluído que as tarefas robóticas dependem da execução de processos ROS. Adicionalmente, foram identificados problemas que ocorrem devido à execução não controlada destes processos. Por este motivo, foi criada uma classe que controla a execução de todos os processos no robô de forma a evitar os problemas identificados, permitindo aos novos programas usarem estes processos através desta classe sem terem de re-implementar a lógica do seu controlo. Após a criação de uma fundação sólida neste capítulo, os capítulos seguintes focar-se-ão no desenvolvimento de novo software.

No terceiro capítulo foram identificadas as tarefas robóticas a implementar neste projeto. As tarefas foram extraídas da documentação de quatro competições robóticas, com o intuito de que se o robô tem capacidade para participar nas competições, então tem necessariamente utilidade para servir as pessoas na universidade. Nem todas as tarefas presentes na documentação foram escolhidas pois requerem ferramentas que o robô não possui de momento, como um braço robótico para manipulação ou um drone. A lista final contém sete tarefas. De seguida, o objetivo de cada tarefa é explicado, bem como a lista de passos que a constituem. Após a identificação das tarefas, foram identificados três conjuntos de funcionalidades que são comuns a grande parte das tarefas. São estes: navegação, a capacidade do robô navegar no seu ambiente de forma autônoma; fala, a capacidade de o robô comunicar verbalmente com os humanos; reconhecimento de fala, a capacidade do robô perceber o que os humanos comunicam de forma verbal. A implementação inicial destas funcionalidades facilita o desenvolvimento das seguintes tarefas, pelo qual foram priorizadas.

O quarto capítulo explica o que é a navegação autônoma e expõe a sua necessidade. De seguida é explicado que a navegação autônoma nos robôs é atingida em três passos: 1) mapeamento, 2) localização, 3) criação de um caminho. O primeiro passo, mapeamento, consiste na criação de um mapa do ambiente onde o robô se deslocará autonomamente. Nesta secção é explicado o algoritmo responsável pela criação deste mapa, SLAM. Adicionalmente, são também expostas e comparadas duas abordagens que o implementam. Na seção seguinte é explicado como efetuar a localização do robô no mapa gerado no passo anterior. O algoritmo, baseado no filtro de partículas, MCL é explicado com algum detalhe. De seguida, as ferramentas que o implementam e consequentemente permitem localizar o robô são apresentadas. Seguidamente, o passo da criação de um caminho que permite que o robô se desloque da sua localização inicial à localização desejada é explicado. Por último, foram criados programas adicionais relacionados com a navegação. Um programa monitoriza a localização do robô em cada instante de forma a que outros programas tenham acesso a esta informação. O segundo permite efetuar a rotação do robô em torno de si mesmo.

O quinto capítulo dedica-se à explicação da implementação das funcionalidades que fornecem a comunicação entre o robô e os humanos. O robô simula fala, de forma a comunicar com os humanos verbalmente, através da conversão de uma frase de texto para um ficheiro áudio. Esta conversão é efetuada através do uso da biblioteca GTTS. O ficheiro de áudio é depois executado, simulando a fala do robô. De forma inversa foi implementada a funcionalidade que permite que o robô perceba a fala dos humanos. Esta funcionalidade foi implementada através do uso de um conjunto de microfones ReSpeaker que capturam o áudio que contém a fala dos humanos. Este áudio é depois enviado para uma biblioteca do ROS que interpreta a fala presente no áudio e o converte em frases de texto. Por último, foram criadas funções que permitem usufruir de ambas as funcionalidades.

O sexto capítulo corresponde à implementação da tarefa intitulada ”Help-me-carry”. Nesta tarefa o robô tem de seguir o humano até que este lhe peça que pare. De seguida o humano coloca um saco em cima do robô e indica-lhe para o entregar numa determinada localização, algo que o robô terá de realizar sem falhas. A tarefa já estava implementada na sua maioria, pois foi o projeto de licenciatura de um colega. Contudo, a comunicação entre robô e humano era efetuada através de códigos QR, algo que teve de ser alterado para a fala e reconhecimento de fala, implementado no capítulo anterior. Adicionalmente, o código foi dividido em módulos para que futuras tarefas possam usufruir das mesmas funcionalidades. Por último, algumas alterações foram efetuadas ao código de forma a torná-lo mais resistente a erros originados pela câmera.

O sétimo capítulo corresponde à implementação da tarefa intitulada ”Restaurant”. Nesta tarefa o robô desempenha o papel de empregado de mesa. Constantemente monitoriza o seu ambiente à espera de clientes que o chamem através de um aceno de mão. De seguida o robô desloca-se à mesa do cliente e interpreta o seu pedido. O robô terá de navegar entre a mesa e o bar de forma a receber e entregar o pedido. A funcionalidade que teve de ser implementada do zero foi a deteção de acenos de mão. Vários métodos foram experimentados até se atingir a funcionalidade final. Esta baseia-se na monitorização do movimento horizontal de mãos que estão perto de caras. Os fatores medidos são o número de mudanças de direção, a sua frequência e a amplitude média dos acenos. Os restantes passos foram implementados diretamente usando as três funcionalidades base implementadas anteriormente. Adicionalmente, especulou-se sobre uma forma de criar um modelo capaz de realizar a deteção de movimentos de mãos. Contudo, devido a restrições temporais resultantes das tentativas falhadas, combinado com o facto de não se saber se o método funcionaria, optou-se pela implementação da funcionalidade descrita anteriormente, ficando esta apenas como potencial candidata a solução de problemas semelhantes.

O oitavo capítulo corresponde à implementação da tarefa intitulada ”Speech, Object and Person Recognition”. Nesta tarefa o robô terá de responder a uma pergunta posta pelo humano e reconhecer um objeto que o humano lhe mostra. De forma a resolver o primeiro problema, duas abordagens foram implementadas. Uma requer internet, usa a API da Wikipédia de forma a procurar a questão e obter a resposta no resultado com maior correspondência. A segunda abordagem foi implementada pois o robô deve ter utilidade mesmo quando não tem acesso à internet. A partir de uma API foram gravadas milhares de perguntas e respostas num ficheiro de texto, onde o robô procurará a pergunta que lhe foi posta. Foi concluído que a segunda abordagem é rudimentar pois será rara a ocasião em que a pergunta se encontra dentro do ficheiro de texto. De seguida foi enfrentado o problema de reconhecimento de objetos. Os objetos requeridos na competição foram identificados e re-criados no laboratório. Por fim, um conjunto de dados com estes objetos foi criado, seguido do treino um modelo capaz de os reconhecer. Devido a estes objetos não serem comuns no dia a dia, um segundo modelo foi treinado num conjunto de dados com objetos mais comuns, de forma a que o robô tenha utilidade fora das competições.

Contudo, os tipos de objetos reconhecidos pelo robô estão ainda limitados pelo conjunto de dados. Este problema é resolvido através da implementação da última tarefa.

O nono capítulo corresponde à implementação da tarefa que requer que o robô use um elevador de forma a mudar de piso. A tarefa requer que o robô entre e saia de forma autónoma no elevador. Contudo, o robô pode ser assistido pelos humanos de forma a escolher o piso bem como identificar quando o elevador chegou ao piso desejado. Esta comunicação foi implementada usando as duas funcionalidades de comunicação entre robô e humano. A implementação da entrada e saída do elevador seguiu uma abordagem dinâmica, que funciona em qualquer elevador. O robô deteta a abertura da porta do elevador usando informação da câmera. De seguida identifica o estado do interior do elevador através do seu mapeamento. Depois identifica a posição mais espaçosa do elevador à qual se deslocará de forma a minimizar a probabilidade de tocar em humanos. A saída do elevador é efetuada de forma semelhante.

O décimo capítulo é dedicado à explicação de duas tarefas, ”Welcoming Visitors” e ”Catering for Granny Annie”. Na primeira, o robô monitora as imagens da câmara e deve reconhecer pessoas que aparecem nas imagens. Para tal efeito foi utilizada uma biblioteca que extrai descritores de caras de imagens. É guardada uma lista de descritores que correspondem às pessoas que o robô conhece. Quando aparece uma pessoa na imagem o robô compara os novos descritores com os da base de dados. Caso não reconheça a pessoa, pede a um humano que a identifique, atualizando depois a base de dados. Na segunda tarefa, o robô deve perceber comandos verbais falados pelos humanos. Na documentação estão especificados três tipos de comandos. Cada um foi implementado através da criação de um autómato que indica se a frase falada pelo humano corresponde ou não ao comando.

O décimo primeiro capítulo é dedicado à explicação da tarefa intitulada ”Getting to Know my Home”. O robô tem um mapa semântico do ambiente, que contém a lista de objetos presentes e a sua respetiva localização no ambiente. De seguida o robô deve percorrer o ambiente e detectar alterações comparando as informações atuais com as informações presentes no mapa. Embora a criação do mapa possa ser feita estáticamente, foi criada uma abordagem para automatizar a sua criação. Adicionalmente, foram realizadas várias experiências de modo a identificar qual o tipo de descritores a ser utilizado para a expansão dinâmica do conhecimento do robô sobre objetos. De seguida, um algoritmo para automatizar o processo de deteção de alterações no ambiente foi criado, com base na funcionalidade de navegação, leitura do mapa semântico e extração dos descritores.

Por último, os principais resultados deste trabalho são resumidos no décimo primeiro capítulo. Para além disso, discutem-se também as principais contribuições para o desenvolvimento da integração de novas funcionalidades. De seguida, faz-se um resumo das implementação das várias tarefas. Finalmente, é exposta uma perspetiva final resultante da realização deste trabalho.

Abstract

Service robots are becoming more commonplace every year due to advances in artificial intelligence, substituting humans in increasingly more complex tasks. By having an autonomous and competent service robot performing routinely tasks instead of its human owners, their productivity increases. The search for better service robots has led to the creation of competitions where such robots are tested and the state of the art technology is pushed further. *Socialab* acquired a *Turtlebot2* robot to serve people around the university campus and one day participate in such competitions. With hopes of achieving these goals, the laboratory has proposed a variety of projects over the years, each adding new layers of functionality to the robot. With each student that has tackled their respective project, the developed software has been continuously stacking. However, each completed project has remained separate from each other and hasn't been used ever since. Hence, the developed software is being wasted. Therefore, it is imperative to integrate all the available software into the robot. Yet, as new projects are proposed, the problem of scattered software can reoccur after the integration of the currently available ones. Furthermore, with more functionality that is developed, the harder and longer it takes to complete their integration. To prevent this entirely, it is necessary to create structural software that eases the development of new functionality as well as its integration with the current software. To achieve this, a class was developed which is responsible for controlling the execution of all processes running in the robot, of which the different software depends on. Additionally, research was done on multiple competitions to identify the most commonly required functionality traits, which we refer to as modules. Afterwards, an implementation of each of these modules was developed. Because of their universality, their implementation allows future software that requires any of the modules to simply import them, rather than having to re-implement them. In line with good software quality practices, if any of the modules needs an upgrade, this upgrade simply has to be performed on the respective module, instead of upgrading every adjacent software that uses this module. This was the first goal of this thesis. After creating a solid foundation for robot software development, the focus shifted towards the creation of new functionality. The different tasks were obtained from the previous research of various robotic competitions. The idea is that if the robot can perform such tasks then it can participate in the competitions, while the same functionalities can be used around campus. The list aimed to be as long as possible with the goal of leaving the robot with as much functionality as possible while taking into consideration the time restraints of the development of this thesis. Seven tasks were selected. The implementation of each task is explained in detail. As each task was developed, the implemented steps were turned into modules, therefore respecting the initial goal of flexible and reusable software. Because of this, as more tasks were developed the following task's implementation was increasingly simpler as some of the requirements were already available from the development of their predecessors. The tasks required knowledge from different areas of artificial intelligence. This lead to the broadening of my knowledge rather than specialization in a single area.

With this work, we show how distinct robotic tasks were implemented. Due to the varied nature of the tasks, we show how to tackle a multitude of different problems that appear in the area of artificial intelligence. Additionally, the work presents an approach to create a solid foundation for the development and integration of increasingly more software. The tasks are benchmarked, meaning future updates of the tasks can be performed and proved superior through the comparison of their results.

Keywords

Robotics, Artificial Intelligence, Machine Learning, Classification, Semantic Segmentation, Detection

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AGI	Artificial General Intelligence
AMCL	Adaptive Monte Carlo Localization
CNN	Convolutional Neural Network
GB	Giga Byte
CPU	Central Processing Unit
GPU	Graphics Processing Unit
GTTS	Google Text-To-Speech
MCL	Monte Carlo Localization
ML	Machine Learning
NN	Neural Network
PC	Point Cloud
PCA	Principal Components Analysis
PCL	Point Cloud Library
PFH	Point Feature Histograms
PFHRGB	Point Feature Histograms RGB
PGM	Portable Gray Map
OGM	Occupancy Grid Map
ORB	Oriented FAST and rotated BRIEF
OS	Operating System
RAM	Random Access Memory
RGB	Red Green Blue
ROS	Robot Operating System
SIFT	Scale-Invariant Feature Transform

SLAM	Simultaneous Location And Mapping
SURF	Speeded Up Robust Features
TF	Transform
YAML	Yet Another Markup Language

Chapter 1

Introduction

With advances in technology, more complex hardware is available. Additionally, continuous advancements in the area of artificial intelligence allow computers to solve more problems which were previously restricted to humans. Because of this, robots can incorporate these software models and, combined with the hardware, substitute humans in increasingly more complex tasks. Furthermore, by having an autonomous and competent service robot performing routinely tasks instead of its human owners, their productivity increases. Therefore, service robots are becoming more commonplace.

The search for better service robots has led to the creation of competitions where such robots are tested and the state of the art technology is pushed further. The state-of-the-art service robots can manoeuvre around the environment autonomously, without colliding with objects and without posing a threat. Additionally, high-end service robots are built in ways that resemble humans. They possess robotic arms which, paired with high-end algorithms, allows them to grasp even the most complex of objects. Advancements in artificial intelligence, specifically machine learning, allow robots to detect and recognize distinct objects. Also, big progress has been done to tackle the problem of speech recognition which in return has lead to the creation of commonplace robots such as Amazon's Alexa. Also, drones are planned to have a big role in the delivery of items such as online orders or emergency medicine.

It is safe to say that service robots will be more frequent in human lives. Because of this, *Socialab* acquired a *Turtlebot2* robot with the purpose of having it serve people around the university campus, as well as one day, participate in robotic competitions. With hopes of achieving these goals, the laboratory has proposed a variety of projects over the years, each adding new layers of functionality to the robot. However, after each student has finished their respective project, the developed software has remained separate from the rest and hasn't been used ever since. Therefore, it is imperative to integrate all the available software into the robot. Yet, as new projects are proposed, the problem of scattered software can reoccur after the integration of the currently available ones. Furthermore, with more functionality that is developed, the harder and longer it takes to complete their integration. To prevent this entirely, it is necessary to create structural software that eases the development of new functionality as well as its integration with the current software.

1.1 Motivation and Objectives

This work has two main goals. The first goal is motivated by the existence of several isolated projects that created new functionality for the robot. This work will not only have to integrate the currently existing functionality but will also have to prevent this problem from reoccurring. Therefore, the first goal is to create a solid foundation of software which eases the development of new functionality into the robot, as well as its integration with the currently available one. Afterwards, the focus of the thesis will then shift into the development of new functionality, aiming to leave the robot with as many capabilities as possible. This goal ties into the first as the development of new functionality is a self-feeding process. In other words, the more functionality is available, the easier it is to develop a new one. This is because certain robotic tasks share steps and requirements. Therefore, if a task requires functionality that is already available, its implementation becomes automatic. Hence, this thesis will aim to develop, not only as much functionality as possible within the time restraints, but also the most common types of functionalities required in robotic tasks.

1.2 Thesis Organization

The remainder of this thesis is organized as follow: Chapter 2 focuses on the creation of a strong software foundation which eases the creation and integration of new functionality. Chapter 3 lists and explains the explored robotic competitions which were used to identify the new robotic functionalities to develop in this thesis. Chapter 4 presents an explanation of how autonomous navigation can be achieved in robots. Furthermore, it explains what navigation-related functionalities were created. Chapter 5 focuses on the communication between the robot and humans. Both ways of communication were implemented and explained. Chapter 6 describes the ”help-me-carry” task. Since it was mostly implemented by a colleague, the chapter focuses on the explanation of the improvements that were made to it, as well as the way the task was integrated. Chapter 7 describes the implementation of the ”Restaurant” task, with a focus on the creation of a method that detects when people handwave. Chapter 8 explains how the ”Speech, Object and Person Recognition” task was implemented, which required the creation of an object recognition model as well as a way for the robot to understand a question and provide an accurate answer. Chapter 9 explains the implementation of the ”Elevator” task, where a dynamic approach was developed which allows the robot to enter and exit any type of elevator. Chapter 10 incorporates the description of two tasks ”Welcoming Visitors” and ”Catering for Granny Annie”. One requires the robot to identify people while the other requires the robot to identify what verbal command was spoken. Chapter 11 describes the implementation of the final task named ”Getting to know my home”. The task requires the robot to identify changes made to the environment. A dynamic approach to obtain the initial information of the environment was created. Finally, conclusions are drawn in chapter 12 and a review of the work the developed work is outlined.

Chapter 2

Foundations

Before being able to work on new software, the robot needs to be properly configured. The configuration of the robot at the start of the project was outdated. To develop the best possible software, it is best to use the latest versions of the libraries and frameworks. Therefore ROS and the necessary ROS packages needed to be updated. Additionally, the computer's operating system was also outdated and, combined with the fact that the newest ROS version was incompatible with the current operating system's version, it was also updated. Also, a few tools were developed to support the future development and integration of new software. This chapter is dedicated to the explanation of what updates were done to the robot's software as well as what tools were developed with the intent of providing a solid foundation for future software development and integration.

2.1 Software Updates

The starting setup was still using ROS Indigo which was released back in 2014. The latest ROS version is Melodic. There are other three ROS versions in between Indigo and Melodic. Needless to say that ROS was heavily outdated and required an update. On the other hand, the computer's operating system, *Ubuntu*, was still on version 14, while the latest version, at the time of this project's development, is 18. This combined with the fact that ROS was only compatible with *Ubuntu* 18 made the decision to update *Ubuntu* unquestionable.

These decisions did, however, come with a few downsides. Unfortunately, *TurtleBot2*'s ROS packages weren't yet released for ROS Melodic. This fact made the installation of these packages more complicated than just using the usual package installer. The packages had to be installed from source. Additionally, changes had to be made to make them work in ROS Melodic.

2.2 Process Controller

Before the explanation of what is the process controller, it is necessary to first understand why it was created. Consequently, to understand why it is necessary, it is required to understand how ROS works. It is worth pointing out beforehand that the following explanation is very general and that ROS has many other features than the ones that will be mentioned. This explanation's purpose is to simply point out problems that led to the creation of the process controller. In a nutshell, the Robot Operating System (ROS) works through the creation of nodes.

There can be different types of programs, each with its different utility for the robot, however, there needs to be a way to integrate the newer programs with the current ones. This is done by making each program be its own node. ROS then provides ways for different nodes to communicate with each other through tools such as topics. All the nodes and the information passed through them are controlled by a master node, *roscore*. Some nodes provide information by publishing it to the topics (publishers), while other nodes may then use this information by subscribing to the topic and reading the information as it is published (subscribers). Because of this, if a program requires a specific type of information, then the program responsible for publishing this information to the topic needs to be running, else the subscriber will not receive any information. For example, if a program wants to have access to the images captured by a *kinect* camera then a process responsible for publishing these images to topics has to be running. Hence we can say that this program is dependent on the execution of a certain process to work. All programs that create a node are dependent on *roscore* to be running as this is the program that registers and keeps track of all the nodes, as well as controls the flow of information. Without it, no nodes can be registered, meaning that no publishers or subscribers can be initialized, hence no communication can occur between programs. Additionally, nodes must be unique, two nodes can not exist with the same name. Meaning that a second process that tries to launch the duplicate node will crash. The solution isn't as simple as just assuring that the same program isn't called again if one instance of it is already running, because a different program may share the same node and therefore still cause the crash. This idea can be understood by analysing *Turtlebot2's 3dsensor* launcher and AMCL's launcher. Both require the use of the *kinect* camera, hence they share some nodes. Consequently, if one is launched while the other is running, one of them will be closed and then the logic of the program that was dependent on the process that was closed will fail. In a nutshell, the problems that need to be addressed are the following:

- Duplicates - A second instance of a program that initiated a node can't run.
- Dependencies - Some processes depend on the execution of others.
- Incompatibilities - Some processes can't be executed at the same time.

These problems can be avoided by considering them every time a program, which needs to launch a process, is created. The program can check if the processes that it depends on are running, if the process is already running and if incompatible processes are running. However, this means that the logic will be rewritten every time a new program that launches a process is created. In the long run, making changes and maintaining the synchronization of all the different modules will become unmanageable. Because of this, it is better to create a centralized program that takes care of launching new processes and handling all the problems. Then, if changes need to be made, they only need to be done on this program, rather than searching for all occurrences of it on all the different programs. We will refer to this program as the process controller.

2.2.1 Implementation

This program keeps track of all the processes that were started through it by storing them in a dictionary. This dictionary maps the process's identification, a unique key, to the actual process.

2.2.1.1 Starting a process

This is the main function of the program. It receives the command string to be executed as well as its identification key. The key's value is somewhat irrelevant but it must be unique in the set of all process keys. The program will then detect if the key exists in its dictionary. If this is not the case then it will start the process and add it to the dictionary. If it does already exist then it will check if the process is still running. If it is running then it won't do anything to avoid duplicates. If on the other hand, the process is not running then it the function will start it. This case corresponds to a process that ran before but either finished or its stop function was called, hence why it exists in the dictionary. The execution of the process is done by using a python library named `subprocess`. A new shell is created where the process will be executed.

This function starts a process by receiving a command. With this logic, the problem of having duplicate processes is avoided. However, the other two types of problems can still occur and must be tackled. Yet, the dependencies depend on the process itself and need to be individually specified. Because of this, each process must have a start function where the processes it depends on are started (by calling their respective start functions).

In order to fix the problem related to incompatibilities, we will compare the nodes that the process would register with the nodes already registered. If the process wishes to register a single node that is already registered, the launching of the process is cancelled. This means we are prioritizing the processes that already running and protecting them from crashes due to incompatibilities. If a process fails to start because of nodes already being registered, then the programmer must change the logic of the programs as it contains conflicts. The list of nodes that are registered at a specific time can be obtained by calling the function `'get_node_names'` from the *rosnode* library present in ROS' Python API. The nodes that the process wishes to register have to be identified in the process' start function. To identify the nodes registered by a specific process, the process needs to be launched and the previous function must be called before and after the launch. The nodes registered by the process are the ones that were not registered before the launch.

Additionally, there is one last problem to take into consideration. The `subprocess` library starts a process in a different thread and returns the control of execution to the main thread immediately after. This means that the next instruction of the main thread may occur before the process is fully operational and, considering that most ROS processes take a few seconds to initialize, this is most often the case. However, when a program calls the function to start the required process it expects the process to be running when the function finishes.

The nodes that the process initializes aren't registered instantly. This fact means that if the program depends on the creation of these nodes and therefore starts the process responsible for their creation but doesn't wait for it to register them, then it will fail or potentially crash. Because of this, the start functions must wait for the process to be fully initialized before returning. This is achieved by waiting for all the nodes that process is expected to register in `roscore` to be registered.

The list of registered nodes is obtained using the function mentioned before. Since we know the nodes that are launched by the process (to avoid incompatibilities), we can simply wait for these to be present in the list. A function was created to achieve this. It receives a list of nodes and only returns when all of them have been registered.

The process controller should provide an interface to start and stop every single type of process. Because of this, if more processes are necessary other than the ones that are already provided with this project, then the new start function should follow the following formula:

1. Have a unique key to identify the process in the dictionary.
2. Specify the nodes that it wishes to register.
3. Check if any of the nodes is already registered and if so stop the launch.
4. Start the processes it depends on by calling their respective start function. All of them have to at least call the start function of *roscore*.
5. Call the start process function to start a process with the respective command and key.
6. Wait for its nodes to be registered.

2.2.1.2 Stopping a process

It is best practice to stop a process after the program that required it no longer needs it. Both for CPU optimization but also because it may prevent the execution of processes that share registered nodes. This function receives the key of a process and checks if it exists in the dictionary. If not, then the process doesn't exist therefore can't be closed. If it exists then it checks if the process is running. If it isn't running then it is already stopped hence nothing needs to be done. If it is running then it stops it. The function waits for the process to close completely before returning rather than just sending the signal to terminate and immediately returning. This way we don't risk starting an incompatible process whilst the other is still 'cleaning up' and consequentially leading to a crash.

2.3 Conclusion

In this chapter we identified that the initial robot setup was heavily outdated. Therefore, both ROS and the operating system, Ubuntu were reinstalled. However, this fact meant that *Turtlebot2*'s packages had to be manually installed which required some tinkering. Secondly. we identified that most programs rely on the execution of ROS processes to obtain external information. However, the simultaneous execution of certain processes leads to fatal errors. Therefore, a program was created which handles the launching, execution and termination of all ROS processes. All of these changes create a stable foundation for the creation of new robotic functionalities.

Chapter 3

Robot Tasks

The goal of the project is to provide the robot with increasingly more functionality to allow it to consequentially solve increasingly more tasks. Then, use the new capabilities to provide utility around campus as well as participate in robotic competitions. Therefore, it becomes necessary to identify and list the functionalities that need to be developed. When it comes to campus utility the possibilities are endless, the only restrictions that do exist have to do with time constraints and the robot's components. The competitions, however, provide a rule book where the tasks that the participants must complete are well documented including goals and restrictions.

By implementing a task for a specific competition, the developed functionality can likely be partially or, in some cases, completely used around campus. Because of this, this chapter will focus on the exploration of possible competitions that the robot can participate in. However, with this said, these are team competitions, they require multiple members because it is impossible for one person to develop all the functionalities required for the robot to compete. Because of this, it is necessary to select tasks to solve in the scope of this project taking into consideration the already available functionality developed by my colleagues. This chapter is dedicated to the explanation of the chosen tasks and the reasons behind each respective choice. At first, a summary of the explored competitions will be provided. Then the tasks that were selected from the competitions will be explained, accompanied by the reasons that led to their choice, as well as justifications as to why other tasks were discarded.

3.1 Competitions

Some research was done on the available competitions considering their quality, challenge and location. RoboCup@Home league [MRW+19] is an annual international competition for autonomous service and assistant robots to compete in. Its main goal is to encourage the development of new technology for personal and domestic applications. Anyone with an autonomous robot can participate. The @Home league consists of a sequence of tests followed by an open challenge to demonstrate the abilities of the robot. To participate in the open challenge the robot must participate in at least one test. The open challenge is composed of two stages each with its own specific set of tasks followed by the final stage where random tasks from the previous stages are selected. The task list changes from year to year as the standards of quality increase. This project references the 2019 competition.

The previous competition corresponds to the end goal of socialab's robotics' work. However, it is better to start low and increasingly tackle bigger challenges. Because of this, the European RoboCup@Home Education Challenge 2019 was also explored. This challenge corresponds to a single stage with a set of tasks. Additionally, SciRoc challenges were explored. These challenges are divided into tests called episodes. The '1ST ERL Smart CIties RObotics Challenge, Milton Keynes, UK, 2019' was the latest European challenge by the time this project was created. This combined set of tasks is where this project's tasks will be selected from.

3.1.1 European RoboCup@Home Education Challenge 2019

This challenge is composed of a set of three tasks. Each task requires the robot to do a sequence of steps which are listed below. Each step tests the robot's capabilities in a specific category. The 2019'S tested categories can be summarized in:

- Navigation - the robot's ability to move around the room, going to specific locations and avoiding obstacles along the way.
- Speech - the ability to verbally communicate with humans.
- Following - the ability to follow a person.
- Speech Recognition - the ability to detected and interpret a person's speech and subsequently perform the correct and desired task.
- Person/Face Detection - the ability to detect who is in the frame.

All three tasks of the challenge were picked and implemented in this project. They are listed and explained in the following sections.

3.1.2 ERL Smart CIties RObotics Challenge

This challenge is divided into episodes each focusing on at least one functionality type. At the moment of the development of this project, the available episodes are the following: 'Deliver coffee shop orders', 'Take the elevator', 'Shopping pick and pack', 'Open the door' and 'Fast delivery of emergency pills'. Only the first two will be attempted as the other three require manipulation to grab bags, open doors and when it comes to the last one, a drone is required. The robot is not a drone and doesn't yet have an arm to perform manipulation therefore these tasks can not be tackled.

3.1.3 ERL Consumer Local Tournament

ERL Consumer [AAA+18] is a competition that challenges robots to perform home activities for humans. This competition's challenges, even though they are set in domestic environments, resemble similar challenges to be posed in smart city environments like shopping malls, such as recognising and picking objects, moving outside the home to populated areas, and/or taking elevators between doors. Therefore, ERL Consumer local tournaments aim to prepare teams for the ERL Smart City Challenges. At the time of developing this project, a tournament is set to happen in Portugal. Because of this, all the required tasks will be attempted so that a university team can hope to participate.

3.1.4 RoboCup@Home league Open Challenge

As mentioned before, this competition is the biggest of its kind, the most competitive and consequently the hardest to participate and be competent in. The chosen task to tackle was the Restaurant task as it is similar to tasks in the previously mentioned challenges, this will be explained in more detail below. Most of the other tasks require manipulation and/or command interpretation. This last one is a problem because there isn't an available list of specific commands and therefore, for one to compete and complete this challenge, a large number of commands would have to be developed which would make a project of its own. Additionally, this type of feature will be implemented to a certain extent with speech recognition for a few tasks.

3.2 The chosen tasks

In this section, all the chosen tasks are explained in detail. The competition from where they were extracted is identified. Then an overall explanation of the task is presented, followed by a detailed list of steps required to complete the task.

3.2.1 Help me Carry

This task was selected from the European RoboCup@Home Education Challenge 2019. The goal of this task is for the robot to help carry bags from one location to another. This task does not test manipulation as the bags are placed on the robot by the human. The main evaluated functionality is navigation because the robot is expected to navigate to the bag's initial location by following a human and then autonomously navigate to the location where the bag must be delivered to. Additionally, speech recognition is evaluated as the robot needs to understand when to stop following the human and where the bag must be delivered. This task was mostly implemented by a colleague during the development of his bachelor's degree project. However, there are missing components in his implementation as well as room for improvement. These will be discussed in the section of the task's implementation.

The ”Help-me-Carry” task is composed of the following steps:

- Setup : The robot starts in a known *home* position. A person waits for the robot in a known location.
- Step 1: The robot goes to the human’s location. This step tests *Navigation*.
- Step 2: The robot speaks indicating it’s ready to follow. This step tests *Speech*.
- Step 3: The robot follows the person. This step tests *Person Detection* and *Navigation*.
- Step 4: The robot listens to the person’s command indicating where to deliver the bag that was just placed on it. This step tests *Speech Recognition*.
- Step 5: The robot autonomously navigates to the specified location. This step tests *Navigation*.
- Step 6: The robot speaks indicating it has arrived. This step tests *Speech*.

3.2.2 Speech, Object and Person Recognition

This task was selected from the European RoboCup@Home Education Challenge 2019. As the name says, the task aims to test the robot’s ability to recognize speech, recognize objects and recognize people. However, the person recognition component of this task is somewhat misnamed as there is no person recognition involved. The robot simply rotates, moves closer to the person that was initially behind it and greets them. This fact will become clearer during the explanation of this task’s implementation because the robot will simply move closer to the closest object in front of it, it is just assumed that the object is a person in this task. An object is then shown to the robot, the robot must recognize it and speak the object’s name. Lastly, the human asks the robot a question as to which the robot must provide the correct answer.
The task can be divided into the following steps:

- Setup: A person, holding an object, stands behind the robot.
- Step 1: The robot turns around facing the human. This step tests *Navigation*.
- Step 2: The robot gets closer to the human. This step tests *Navigation* and mainly *Person Detection*.
- Step 3: The robot speaks, greeting the human. This step tests *Speech*.
- Step 4: The person asks the robot a question. The robot must answer accordingly. This step tests both *Speech Recognition* and *Speech*.
- Step 5: The human shows the object to the robot. The robot must identify the object correctly and speak its name. This step tests *Object Detection and Recognition* and *Speech*.

3.2.3 Restaurant

This task was selected because, not only does it not require manipulation or any other type of unavailable hardware, but also because some work has been done as it was part of a colleague's bachelor's degree project. However, he barely got started as it wasn't his main focus, nonetheless, some useful components are available which will help. Additionally, this task is present, with minor differences, in all three competitions. This means that, by implementing it, we essentially implemented three tasks in one. The goal of the task is to create a table serving robot. The robot needs to note and deliver food orders from/to clients in the room. The task can be divided into the following steps:

- Setup: The robot starts in a known *home* position, named the bar. A person, that performs the role of a barman, also awaits in that location.
- Step 1: The robot frequently scans for people waving or verbally calling for it. This step tests *Person recognition* (and waving) on top of *SpeechRecognition*.
- Step 2: The robot must then move to the table where the person, that called for it, is sitting. This step tests *Navigation*.
- Step 3: The robot then listens to the person's order and notes it down. This step tests *Speech Recognition*.
- Step 4: The robot goes back to the *home* position. This step tests *Navigation*.
- Step 5: The robot speaks the order to the barman. This step tests *Speech*.
- Step 6: The robot returns to the table with the order that has been placed on it by the barman. This step tests *Navigation*.
- Step 7: The robot returns to the *home* position and repeats the cycle.

3.2.4 Take the Elevator

This task was selected from the ERL Smart CIties RObotics Challenge episodes. It consists of providing the robot with the capability of taking an elevator semi-autonomously. The robot must navigate autonomously in and out of the elevator but can rely on human-assistance to control the elevator and identify the elevator's current floor. The implementation of this task comes with the additional benefit of allowing the robot to move from floor to floor in campus buildings by taking the elevator.

The ”Take the Elevator” task can be divided into the following steps:

- Setup: The robot waits in front of an elevator with people inside.
- Step 1: The robot enters the elevator after detecting the opening of the door. This step tests *Navigation*.
- Step 2: Robot speaks to the humans asking them to press a floor number. This step tests *Speech*.
- Step 4: The robot waits for humans to indicate that the elevator has reached the desired floor. This step tests *Speech Recognition*.
- Step 5: Robot exits the elevator. This step tests *Navigation*.

3.2.5 Catering for Granny

This task, as well as the following two tasks, belong to ERL's Consumer Local competition. In this task, the robot has to understand a set of fixed commands. After understanding the command the robot will have to perform the corresponding task. The list of commands is the following:

- Turn on/off the lights - the robot has remote access to a room's light controls. It can then turn the lights on or off.
- Lower the blinds - the robot has remote access to the blinds controls. It can then lower or raise them.
- Find and grab an object - the robot is requested to fetch an object. The robot must understand which object was requested. It will then have to find it, grab it and deliver it. The grabbing part will have to be ignored for the time being as the robot doesn't have an arm yet.

3.2.6 Welcoming visitors

The robot has access to a surveillance camera. When someone knocks on the door or rings the bell the robot will have to identify the person. Depending on who the person is the robot will then perform a specific task. For example, if the person is the mailman then the robot would open the door and accept the letter(s).

3.2.7 Getting to Know My Home

In this task, the robot will have to create a map of the room. With it, the robot should be able to navigate the environment. Then, objects will be pointed to and named. The robot will have to understand this behaviour and memorize the object. Finally, the robot must autonomously navigate the room and identify changes that occurred since the learning phase. These changes include objects moved from their initial location and new objects placed in the room.

3.3 Conclusion

The goal of this chapter was to identify what features should be implemented by the end of the thesis. To achieve this, research was done on multiple robotic competitions. By implementing these tasks we simultaneously provide functionality that allows the robot to participate in such competitions while also gaining some utility for campus usage. Seven different tasks were selected. After selecting the individual tasks five primary modules were identified. These are Navigation, Speech, Speech Recognition, Object Detection. These functionalities correspond to the most common requirements in robotic tasks.

Chapter 4

Navigation

Most of the selected tasks require the robot to autonomously navigate its environment to achieve the goals. However, without care, the robot will either not know where to go or crash in its path. Because of this, a proper navigation module needs to be implemented to allow the robot to move around the environment safely and effectively. This chapter will focus on the navigation module explaining how it works and how it was implemented in our robot. In a nutshell, for the robot to be able to navigate around the environment, a map of said environment needs to be developed. Then, the robot needs to know in which location of the map it is at. Lastly, a path to the desired location needs to be created. All these components will be explained in the following sections.

4.1 Mapping

This step involves creating a map of the environment the robot will navigate in. This is required because if the robot were to navigate blind, the robot wouldn't know where in the environment it is currently standing, therefore it wouldn't know in which direction it should move to go to the desired location. A simple analogy can be made by comparing the robot with someone that decided to travel to a place they have never been before and have just landed in the airport. How does the person know which direction to go? Well, the answer is by using a map.

4.1.1 Requirements

The map is created by moving the robot around the environment, allowing its sensors to capture information about obstacles in it. To achieve this goal, there are a few requirements. Besides having a mobile robot, it is also necessary to have some kind of hardware to act as sensors to capture information about the environment. In this project, a *Microsoft XBOX 360 Kinect* camera was used to perform this task. Afterwards, an algorithm to create a map using the information of the sensors is required. ROS offers many implementations of the SLAM algorithm to perform this task. This algorithm will be explained in more detail below. Additionally, it is useful to use a tool that allows the visualization and control of this process, the tool *RViz* was used for this.

4.1.2 Simultaneous Location And Mapping (SLAM)

SLAM is the algorithm used to generate a map of the room. By reading its name one can deduce that it also locates the robot in the map that it is generating simultaneously. It achieves this by reading the data of the robot's sensors, more specifically the depth data, gathering a series of 3D views of the environment with approximate positions and distances, as well as odometry data to know the rotation of the robot's wheels for localization. The two most popular SLAM approaches are GMapping and RTAB-Map. Both generate very similar maps, meaning they have similar performances in this regard. They both implement their version of the SLAM algorithm, therefore they both convert the information of the sensors into a map similarly. However, there is one big downside to GMapping: the generated map is static, in other words, once an area of the map is drawn it cannot be changed. This means that if the room changes, for example by placing a chair in a previously empty position, this chair will not be shown on the map and, consequently, the robot may crash into it. On the other hand, RTAB-Map updates the map with the most recent information of the room. Using the previous example this means that if the map generates an empty position but later a chair is placed there, the algorithm will detect it and update the map. Because of this, the latter was chosen in this project as the SLAM approach to generate the map of the room.

4.1.3 Map

The map generated by the SLAM algorithm is a two-dimensional Occupancy Grid Map (OGM). It is essentially an image, hence the grid part. Where each pixel in the grid stores the value of occupancy of that specific place in the map. There are three types of occupancy:

- **Unknown** - represented by the value -1 (negative one). Visually it corresponds to the colour grey. It means the robot doesn't know what exists in this area because the sensors didn't capture any information from it. Either because the map is incomplete or because this area is unreachable. For example, an area outside of a closed room is unreachable and therefore, it is marked as unknown.
- **Empty** - represented by the value 0 (zero). Visually it corresponds to the colour white. It means that the robot didn't detect any obstacles in that location, therefore a path can be created passing through it.
- **Blocked** - represented by the value 100 (one hundred). Visually it corresponds to the colour black. It means that the robot detected an obstacle in that location, therefore the planner can not create a path through it.

The map can be obtained by using a service like *map_saver* available in the package *map_server*. It returns two files: a PGM and a YAML file. The first one corresponds to the actual OGM while the second provides information about the map such as the name of the corresponding OGM map file and it's resolution. This file will be further explained later. The map generation can be visualized using *RViz* by adding a map tool and linking it to the topic where the information about the map is being published to.

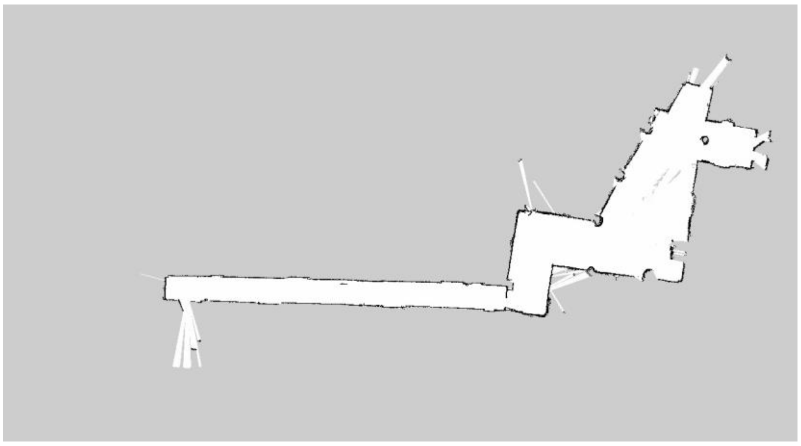

Figure 4.1: Map of the department obtained using SLAM

4.2 Localization

Localization corresponds to the next step in the chain to achieve navigation. It is the process of locating the robot in the previously generated map. Continuing with the earlier analogy, when someone arrives in the airport, in order to be able to navigate around the city, they need a map and to locate the airport (their current position) on the map before moving. Otherwise, they would just walk around randomly not knowing if they are getting closer to their desired destination or farther. This section tackles this problem.

4.2.1 Requirements

The requirements to locate the robot in the map, besides having a map, are similar to the mapping process. The difference is that instead of using a mapping algorithm, a localization one is required. The localization algorithm uses the information captured by the sensors to determine where the robot might be on the map. The algorithm used to perform localization was MCL and is explained in more detail in the next section. Once again *RViz* provides a few helpful tools, this time to visualize the localization process.

4.2.2 Monte Carlo Localization (MCL)

This is the algorithm used to locate the robot in the map generated during the mapping phase. To achieve this, the algorithm generates a set of possible particles, sometimes also referred to as poses. Hence, the algorithm is also called a particle filter. Each particle represents a possible position and orientation of the robot on the map.

Similarly to the mapping process, as the robot moves around the room, it collects more information about its environment. By using this information, more and more particles that are wrong, are discarded. Over time the remaining particles will concentrate on a spot. This algorithm is used by the *'amcl'* node that comes in the ROS package of the same name. This node requires information from the sensors and from the map therefore it subscribes to their respective topics. The information about the current possible particles are published into the topic named `'/particlecloud'`. *RViz* allows the visualization of these particles, each red arrow in the image 4.2 corresponds to a possible position.

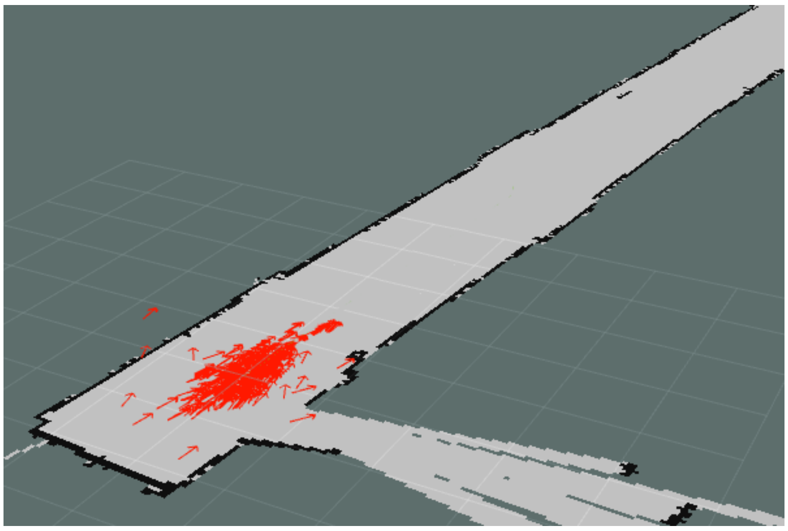

Figure 4.2: Simulation of localization in RViz

The image represents a typical state of the localization process before the robot has moved around the environment and collected enough information to precisely estimate its position. The arrows are already somewhat congregated due to two reasons. First, even though the robot is stationary, it is already receiving information about the part of the environment in front of it and in range of the sensor. However, the main contributor to this is the estimation of an initial pose. This estimation can be done by hand through *RViz* or, most usefully, by specifying the pose parameter to AMCL at its launch. If this parameter isn't specified then its values reset to zero.
The robot's current estimated location can be obtained by reading the information published to the ROS topic named `'/amcl_pose'`.

4.3 Path Planning

This step consists of the creation of a path that leads the robot from its current location to its desired location. Therefore, this step requires an occupancy map, localizing the robot in the map and a goal location. In other words, all the steps explained before built up to this. It is important to note that perfect localization isn't required to create a path. As the robot moves along the path, the localization algorithm keeps discarding the invalid particles consequently adjusting the estimate location.

However, the poorer the precision of the initial localization the more adjustments will have to be performed to the path until the goal is reached. A path is created by sending a goal pose (position and orientation) to a ROS service named `MoveBaseGoal`. The algorithm will create a sequence of checkpoint positions that lead to the final position. To reach each checkpoint, the algorithm will apply velocity to the robot with a direction based on the current position and the next checkpoint. The information about the calculated path can be obtained by subscribing to ROS topic named `'MoveBase/plan'`. Once again, *RViz* provides a tool which allows the creation of a navigation goal as well as the visualization of the current path. In figure 4.3, an example is presented of *RViz* visualizing a path after a navigation goal has been established. The path is represented by the green line.

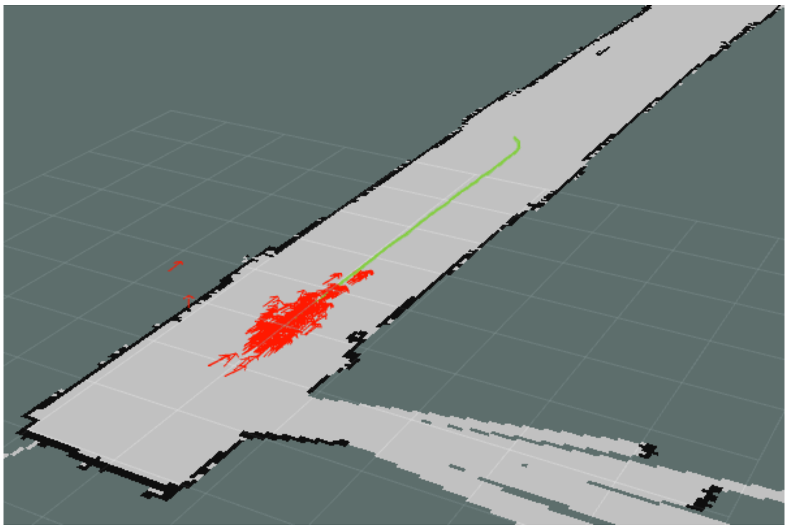
Figure 4.3: Simulation of path planning in RViz

4.4 Developed Sub-Modules

The first half of this chapter was dedicated to the explanation of how to use ROS's tools to build a strong foundation for the robot to perform navigation. This section will focus on the explanation of what functionalities, that are related to navigation, were developed and added to the robot.

4.4.1 Pose Tracker

Many tasks need to know the robot's current position and/or rotation. So instead of having to keep re-writing the code as well as potentially, because of multi-threading, having the robot asking for its position on two different modules at the same time, a sub-module was created solely for this purpose. This feature will keep a global variable that corresponds to the robot's current pose. This variable is updated frequently and can be accessed by all the other modules that need it. A subscriber is created for the `/amcl_pose` topic that is responsible for keeping track of the robot's pose. Every time a callback is received, the global variable is updated with the current pose of the robot.

4.4.2 Rotation

The Speech, Object and Person Recognition task has a step that requires the robot to rotate 180º degrees (around its vertical axis). Because of this fact, this task was the main motivator for the development of this functionality. However, since rotation is such a general and useful behaviour its implementation will provide a simple and flexible interface so that other modules can easily use it to rotate the robot. This means that instead of just developing a feature that rotates the robot 180º, to achieve the requirements for the previously mentioned task, the rotation sub-module will instead allow the callers to specify the rotation angle. Additionally, to allow for even more flexibility, the callers can also choose between rotating the robot a certain amount of degrees or rotating the robot to achieve a certain rotation value in regards to its initial reference point. The difference between these two is explained below.

4.4.2.1 Implementation

When the TURTLEBOT2 robot is initiated it can be moved around by publishing velocity messages to certain ROS topics. These messages are divided into two components: linear and angular. If the linear values aren't null then the robot will move in the provided direction. On the other hand, if the angular values aren't null then the robot will rotate. Many ROS topics achieve this and they may vary from robot to robot. For this project the chosen topic was `/mobile_base/commands/velocity`. Now that we've figured out how to rotate the robot, the problem that emerges is how to know how many angles the robot has rotated when provided with an angular velocity. To solve this, we'll make use of the previously developed sub-module that keeps track of the robot's current pose. As mentioned before, this sub-module keeps track of the robots current pose which contains information about the robot's current position and current rotation values. Therefore, by reading these values, that keep being updated, we know what was the robot's initial rotation and what is its current rotation as it is being rotated by publishing messages with angular velocity to the topic. Consequently, when its current rotation hits the desired value, we stop rotating the robot by publishing a velocity message with null values to the topic, achieving our goal.

The pose's rotation values are in quaternions, the angular velocity accepts radians hence a conversion had to be done. ROS' transform package provides a function to do this conversion. Additionally, because it is more intuitive to work with degrees rather than radians, the function will accept degrees as its inputs. Therefore, a conversion between these two units was implemented. The angular velocity to provide the robot with at a given moment is calculated based on the difference between the robot's current rotation value and the goal rotation value. The farther away the robot is from its goal, the faster it will move and vice-versa. The robot won't achieve dangerous velocity values because the rotation angle will be normalized between -180º and 180º. This range corresponds to ROS' rotation range.

The robot will stop rotating when its current rotation value corresponds to the goal rotation. However, since the robot may take a long time, or never manage to hit the perfect rotation, as well as the fact that rotating 179.6º instead of 180º is acceptable, a configurable parameter is provided. This parameter corresponds to an acceptable error threshold. As mentioned in the beginning, two functions were created: one that rotates the robot a certain amount of degrees and another that rotates the robot to achieve a certain rotation value. The latter's implementation corresponds to the previous explanation. The first simply adds the desired rotation that it receives from its input to the robots current rotation, obtaining a goal rotation, and passes it to its cousin function, this way no code is re-written.

4.5 Conclusion

In this chapter, we exposed the utility that autonomous navigation provides to a robot. Then, we concluded that to reach this goal there are three requirements: mapping, localization and path planning. Each of these requirements was studied, explained and implemented. Afterwards, functionalities related to navigation were developed and their implementation described.

Chapter 5

Human-Robot Communication

This chapter explains how both speech and speech recognition modules were implemented. The communication of the human with the robot and of the robot with the human is different, however, since they are intrinsically connected they were grouped into a single chapter. In a nutshell, the robot communicates with the human using a text-to-speech library and the human communicates with the robot by the means of a microphone and software that converts speech into text.

5.1 Text-to-Speech

This section explains how the communication of the robot with the human was implemented. This communication corresponds to the implementation of the speech module required in the 'Robot Tasks' chapter. The chosen approach was to use a library that converts text into audio. This audio will then be played simulating speech. Google Text- To-Speech (GTTS) library was chosen for this job. It is a free to use python library that interfaces with google's text-to-speech API meaning it is a perfect fit for our problem. It provides the ability to generate audio files where an input text is 'spoken'. Lastly, a way of playing this audio was required. To achieve this python's Operating System (OS) library was used. Below is a code snippet of the implemented speech module.

```
def speak(text):
    language = "en"
    file_name = '/tmp/output.mp3'
    output = gTTS(text=text, lang=language, slow=False)
    output.save(file_name)  # create file
    os.system("play " + file_name)  #play file
```

The GTTS library is used to generate the speech file where the requested text is spoken. This file is saved in the temporary directory as it's utility ends with this function's execution. Then the audio is played using the OS library.

5.2 Speech-to-Text

This section explains how the communication of the human with the robot was implemented. This communication corresponds to the speech-recognition module required in the 'Robot Tasks' chapter. The chosen approach was to use a 'ReSpeaker Mic Array V2.0'. The ReSpeaker V2.0 is based on XMOS's XVF-3000. This chipset includes many voice recognition algorithms to assist in performance.

Additionally, there is a ROS package developed to interact with this microphone, named `respeaker_ros`. It provides a service that when launched publishes the microphone's information, such as sound direction and localization, into each respective topic. Furthermore, there is another useful ROS package called `ros_speech_recognition` that takes the microphone's audio and converts the speech into text form.
In order to obtain information provided by the microphone one simply has to launch `respeaker_ros` launch file and create a subscriber to the respective topic. Such behaviour is explained in the code snippet below:

```
def callback_ speech_ to_ text ( data ) :
    print ( " Speech=" + str ( data ) )
def  callback_sound_direction ( data ):
    print ( "Direction=" + str ( data ))
rospy . Subscriber ( "/sound_direction" , Int32 ,  callback_sound_direction)
rospy . Subscriber ( "/speech_to_text" , String ,  callback_speech_to_text)
```

In the previous example two subscribers were created, one receives speech data and the other receives audio direction data. Alternatively, if one is only interested in receiving the audio's speech, one simply needs to implement a *'SpeechRecognitionClient'* that frequently publishes the detected speech in the last few time frames. This is exemplified in the code snippet bellow:

```
client = Speech Recognition Client ( )
while True :
    result = client . recognize ( )
    print ( result )
```

5.2.1 Speech-Recognition Functions

Based on the previous logic, three functions were created to fill the different needs related to speech-recognition. The first is a function that returns the first string of speech, recognized by the speech to text algorithm after its call. This function is useful for when the robot accepts multiple answers from the human and needs to identify which one was spoken in order to perform the task associated with each answer. The remaining two functions receive a string as input and only return the execution control when this string is present in the recognized speech. One returns when the recognized sentence is exactly the same as the input string, while the other returns when the string is present in the recognized sentence. The functions are useful for when the robot waits for the human to speak a single and specific sentence or word.

5.3 Conclusion

This chapter explained the implementation of the two communication-related modules: speech and speech recognition. The first is used to allow the robot to verbally communicate with humans. The implementation is based on the use of a text-to-speech library. A function was created which receives as input the text to be spoken by the robot. To perform speech recognition, a ReSpeaker microphone array was used to capture audio. The audio stream is then passed to a speech-recognition algorithm present in an open-source ROS package with the same name, which converts it into text. Lastly, three functions were created that fill the different needs related to speech-recognition.

Chapter 6

Help-me-Carry

This chapter will focus on the explanation of how the 'help-me-carry task' was implemented. This task was initially implemented by a colleague named António Fernandes [FER18] for his bachelor's degree project. Therefore, most of the work was already done and available. However, some changes were required. Ideally, the robot is controlled by voice, not only is this a requirement in the competitions, but it is also more practical than his approach which was to use QR codes. Rather than a person carrying all the QR codes and showing the correct one for the current step, the person will just have to speak to the robot. Additionally, the code was structured in a state-machine architecture which was changed for two reasons. On one hand, it is quite useless as each state of the machine can just be replaced by an ordinary function call. On the other hand, the code is static, each step is used in the task but can't be used in other tasks. This fact was fine in António's project because he only needed to develop this one task. However, the goal of this project is to develop modular code to be reused in other tasks if necessary. Therefore, the code was segmented into new modules in addition to the previously developed ones in this project.
In short, two new modules were obtained from Antonio's code:

- Move to Human - Calling this module will make the robot get closer to the nearest object in front of it. The distance can be tuned in a parameter. This module is important for the 'follow the human' module because the robot will follow the closest object which is guaranteed by this step.
- Follow Human - This module calls the 'TurtleBot_Follow' package which in result makes the robot follow the closest object in front of it. Therefore, if the human is the closest object, the robot will follow him around.

The changes made to the initial implementation of the task can be summarized into:

- Code was segmented - as mentioned before two functionalities were obtained from Antonio's code and transformed into modules. This change makes it so that other programs (or tasks) can just call the functions that implement them, rather than having to rewrite the entire code.
- Code was cleaned and optimized - the state-machine architecture was removed and certain code instructions were removed.

- Move closer logic - The robot moves closer to the human by subscribing to a topic which receives depth data from the camera. This data comes in the form of a matrix of depth values. The robot then obtains the distance to the closest object by finding the centre value. However, this implementation is susceptible to errors. On one hand, the method only accounts for a single value. If the human isn't centred in the camera's frame then the centre value will not correspond to the distance to the human. Additionally, if this specific value is wrong, the distance to the closest object will consequently be wrong. One the other hand, the data published by the camera can contain *NaN* (not a number) values. These can occur when there are no objects in the range of the camera's depth range. Therefore, if the centre value is *NaN* then the program fails. This problem occurred quite frequently which lead to its identification. To prevent all these problems from happening the robot will obtain the distance to the closest object by calculating the mean of values around the centre point of the matrix. The calculation of the mean will first identify and remove *NaN* values from the equation. The size of the range around the centre point is a configurable parameter. From testing the size of the range isn't very restrictive in precision and a range of 20 cells proved to consistentlywork.

- Speech recognition instead of QR codes - the robot will move from step to step either automatically or, if waiting for a human command, by recognizing speech using the previously developed speech recognition module.

- Speech for feedback - the robot will speak to the human giving him feedback about the state of the task for the human to know if the robot understood his previous command and to understand what do robot is currently doing or will do next. This will be clearer bellow.

The resulting task's functionality can be summarized in the following steps:

1. The robot waits for the human to say 'help me carry'.

2. The robot calls the 'move closer' module to get closer to the human trying to reach the distance specified by the customizable parameter, which by default is 20 centimetres. This is necessary because the 'Turtlebot_follower' package will cause the robot to follow the closest object. Therefore by making the robot get closer to the human we assure that the closest object is the human and consequently guarantee that the robot will follow him.

3. The robot speaks informing the human that its ready to follow by speaking 'Ready to follow'. It calls the 'follow human' module.

4. During this process the robot is listening for human commands and waits specifically for the human to say 'stop'. This means that they have reached the desired destination and therefore the robot will stop following the human.

5. The robot will inform the human that it will no longer follow by speaking 'I will stop following'.

6. The robot waits for the human to place the bag that will be delivered. It specifically waits for the verbal command ’go’.
7. Once the robot hears the command, it will navigate to the specified location in the map where the bag should be delivered to. This is done through the navigation module. Therefore, a map of the environment must be created and specified a priori. The robot uses AMCL to localize itself on the map and to move to the goal location. This also implies that the coordinates associated with the spoken location are available in a variable.

6.1 Conclusion

This chapter explained the implementation of the first task, named ’help-me-carry’. The task was initially developed by a colleague, however, his implementation left room for improvement. In the initial implementation, the robot progressed through the task by the means of QR codes. This meant that the human would have to carry all the required QR codes as well as find and show the correct one for the current step. This mechanic was changed to the recognition of speech by the means of the implemented module. Additionally, the code was cleaned and feedback is provided through the task by allowing the robot to communicate its state through the speech module. Then, some changes were performed on the algorithm which brings the robot closer to the human. Lastly, the individual parts of the algorithm were extracted and transformed into individual modules, allowing them to be called not only by this task but also to any other program that needs them without requiring the re-writing of the code.

Chapter 7

Restaurant

This task is common to both SciRoc and RoboCup@Home competitions making its implementation that much more valuable. This section is dedicated to the explanation of how each step of the task was implemented. This task makes use of a handwaving detection module that was created specifically for it as well as other general modules such as navigation, speech and speech recognition.
The task's functionality can be summarized in the following steps:

1. The robot uses the handwaving detection module to detect people handwaving at it.
2. A conversion from camera coordinates to map coordinates is applied, obtaining the position of the table where the person that wove at the robot is located on the map.
3. The robot moves to the table's coordinates.
4. The robot interprets the clients' speeches using the speech recognition module and notes down the order.
5. The robot moves back to the bar and speaks the order that it received.
6. The robot waits for a verbal command "deliver" indicating that the order's items have been placed on top of it.
7. The robot goes back to the table, using the same coordinates.
8. The robot waits for a verbal confirmation "thank you" that indicates that the clients have grabbed their food.
9. The robot goes back to the bar repeating the process all over again.

7.1 Handwaving Recognition

The robot can detect hands and faces in a specific frame through the means of cascade models. However, detecting handwaving implies that the movement of the hand over a set of consecutive frames is analysed. To solve this problem, a few strategies were attempted before arriving at a working solution. Initially, the idea was to create a dataset of some sort that described the movement of the hand when a person is and isn't waving it, then training a model to classify these movements. However, over time, with the occurrence of more and more problems and with the emergence of new ideas, the strategies were modified and the need for a dataset was removed. This chapter will start with the explanation of the attempted strategies, why they failed and how they led to the final algorithm. Afterwards, the final solution is explained in detail.

7.1.1 Attempted strategies

During the process of discovering the best method to detect handwaving, a few strategies were attempted. Though these strategies didn't work, it is important to mention them as they were the reason as to why some techniques were discarded and others used. In this section, all of the attempted and failed strategies will be exposed as well as the reasons as to why they failed.

7.1.1.1 Intensity Matrix Approach

Detecting handwaving implies analysing the movement of the hand over a set of frames. The initial idea was to create a dataset, where the data would describe the movement of the hand in some way. With this dataset, a model would be trained and ideally, it would be able to determine if a specific movement corresponded to a handwave or not. Additionally, the position of the face in the frame was also considered for a couple of reasons: on one hand, the task requires that the 'clients' of the restaurant must be looking directly at the robot whenever they handwave at it, on the other hand, even if the previous requirement did not exist, a person's hand is relatively close to their face when handwaving, this relative position can be useful to separate handwaving from other types of hand movements.

This first strategy attempted to combine a whole set of frames in a single matrix. This matrix would describe the movement of the hand and face during the set of frames to the classifier. To achieve this the hand and face detectors were applied on each frame of the set, obtaining the position and dimensions of the hands and faces existing on each of the respective frames. Afterwards, the centroids of all the hands and faces were calculated giving the location (pixel) of the respective hand or face in the frame. Then, an empty matrix (all of its values are set to zero) with the same height and width as the frames was created. Since the Kinect camera captures images with a size of 640 x 480 this was also the size of each matrix. Subsequently, the values of the matrix in the previously calculated centroid's positions (same row and column) were changed to describe the position of the hands and faces over the set of frames. The intensity of these values increases with the index of the frame and with the index of the hand/face in the frame as there can be multiple hands and faces in the frame. This way no value in the matrix is the same. The hope was that, with the increase of value with frames, the training framework could detect the sequence pattern by figuring out that the higher the value the later in the set it occurred. Also, it was thought to separate the values of the faces from the ones of the hands while also keeping their correlation in the frame. In other words, tell the classifier that the value is from a hand or a face, but encoding the values in a way that the classifier could tell that a face pixel and a hand pixel occurred at the same time (same frame). This was achieved by encoding the values, adding 1000 to value of the hand to obtain the value of the face. This way the values of the hands are bellow 1000 and the faces above it, separating the two, but the value in the 'tens' which indicates the frame, is the same between the face and the hand.

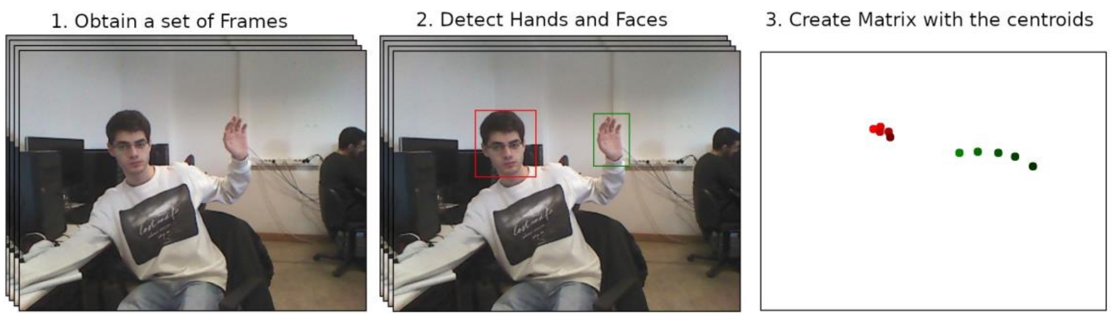
1. Obtain a set of Frames
2. Detect Hands and Faces
3. Create Matrix with the centroids

Figure 7.1: Example of how one intensity matrix is obtained

A dataset with 2208 images was created using the Kinect camera to capture the scene. The images captured by the camera were published to ROS topics. `Rosbag` was used to obtain the sequence of images published to these topics. After applying the intensity matrix method and using a size of 10 frames per set a matrix dataset was obtained containing 2098 entries.

Since each matrix is 640 x 480, this means the problem is 307200 dimensional. Consequently, 2208 representatives of the space are not enough to achieve decent density. Because of this PCA was used to reduce the dimension of the space. The top 20 components were selected corresponding to a loss of information of less than 2%.

Lastly, the data was separated into a test set and a training set. 10% of the data was used for testing leaving the remaining 90% for training. The two sets were created while keeping the percentage of the two classes. 10 different types of classifiers were used for training. Since some classifiers use some kind of randomness, such as initial weight values in a neural network, the results may vary. Because of this, each classifier was trained 10 times. The average scores for the model obtained by training each classifier 10 times can be seen in figure 7.2.

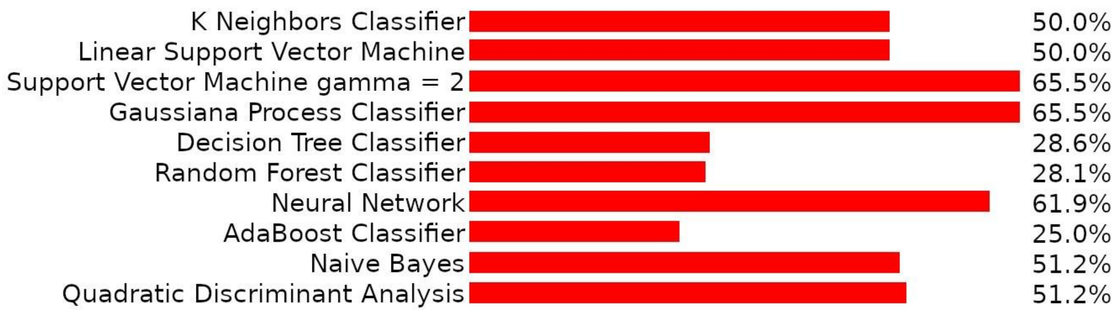

Figure 7.2: Accuracy of different machine learning methods on intensity matrix test set

After observing the results of the previous figure one can conclude that they aren't good enough. Still, to avoid any doubts, the models were tested in a real environment. A program was developed to store the last 10 frames, generate the intensity matrix and pass it to the models. The result was clear, the models would never detect handwaving and would always guess 'not waving'. It's pretty clear after attempting this solution why it failed.

The generated matrix has 307200 values and only a handful are different than 0. Additionally, the values of the hands and faces will rarely be in the same pixel because the person may be closer or farther away from the camera, may be a taller or a shorter person, standing or sitting and so on. This means that, for this strategy to have any chance of working, one would need to generate a dataset of millions of images to generate all possible intensity matrixes for this handwaving problem. To add insult to injury it was also identified that the hand and face detection frameworks sometimes fail to detect hands and faces in cases where these are present in the image. This mainly happens when the hand and/or face was moving very fast and the camera captured motion blur. This consequently results in even less density of hand and face points in the matrix. Needless to say, a better approach was necessary.

7.1.1.2 Position Difference Approach

This approach attempts to fix some of the problems identified with the previous one. Mainly it attempts to solve the problem that the previous one had with its matrix. The matrix had the same size as the image, therefore it had 307200 values, additionally only a handful of them were different than 0. This meant that only less than a percent of the information of the matrix was useful to the classifier. Consequently, all of the classifiers would disregard these useful values and always guess 'Not waving'. This new approach attempts to correct this by only providing the classifier with useful information. In short, not only will this new approach reduce the dimensionality of the problem but it will also increase the density of useful information.

Before moving on, it is important to mention that this approach was not implemented and therefore not tested. It didn't pass the idea stage because of a problem identified with the generation of the dataset combined with the emergence of new ideas of which lead to the final solution. As mentioned before when handwaving, the movement of the hand may be too fast, which leads to the Kinect capturing motion blur which consequently means that the hand detector will not be able to detect hands in the frame making it not just useless but parasitic because the dataset will say that the sequence corresponds to a handwave but the positions of the hands weren't detected and this behaviour is usually associated with 'not handwaving'. A solution to this problem was proposed by just skipping the frames that don't contain any hands. An additional problem stems from the fact that, in some cases, more than one hand and/or face can be in the frame. In a restaurant environment, multiple people enjoy their meals together at the same table. This problem was fixed in the final solution and that method could also be used here to generate a dataset with only the main hand and face. During the brainstorming of this idea, other ideas came up that lead to the final solution and therefore this approach was scrapped.

With this said, this approach is worth explaining because, assuming it works, can lead to the creation of detectors for all types of hand movements, not just handwaving detectors. The idea started with just saving the position of the hand and face in each frame of the set. Since each position can be represented with the x and y value of the pixel each frame would be represented by 4 values. If the size of the set is 10 then each sequence can be represented by 40 values. This process is explained in the figure 7.3.

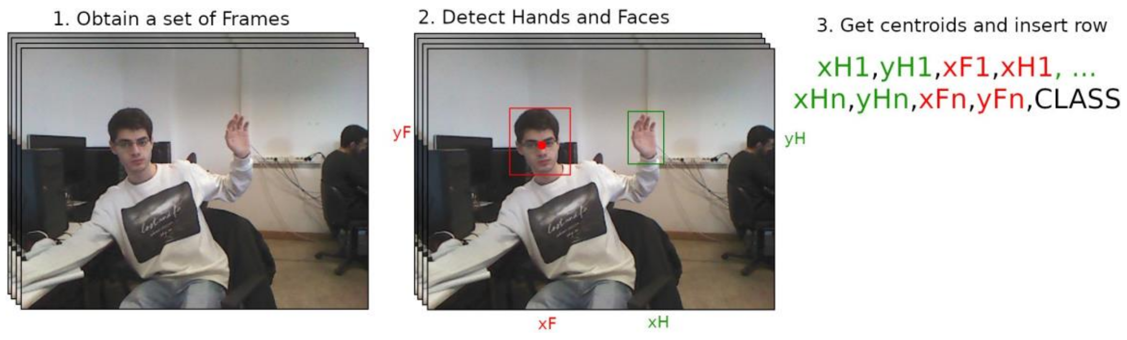
1. Obtain a set of Frames
2. Detect Hands and Faces
3. Get centroids and insert row
xH1,yH1,xF1,xH1, ...
xHn,yHn,xFn,yFn,CLASS
yF
yH
xF
xH

Figure 7.3: Centroids of hands and faces

After some brainstorming a problem was identified: the hand movement can happen in different regions of the camera. In other words, on one moment a person can handwave while standing and their hand will be close to the top of the image while in another moment they can be handwaving while sitting and their hand is closer to the bottom. In short, some sort of normalization is required. The first idea was to subtract the minimum x and minimum y in the sequence to all of the values. This would mean that each sequence would have a hand with a position of 0,0 and all the other ones would be relative to it.

However, there is a flaw in this normalization approach. To normalize a sequence, the minimum and maximum values of x and y are required. In the next sequence, all the images except the first are the same, however, the new image may be the new minimum and therefore the previous common images will be normalized differently in the new sequence even though they are the same points.

Because of this, the process was modified to extract the difference of the position of the hand in consecutive frames. This way regardless of where the hand starts in the sequence, the movement described with this approach should be correct. This idea is exemplified in figure 7.4.

This approach, assuming it works, can be expanded to generate a dataset for different types of hand movements. With such a dataset, one would just have to repeat the process of the previous approach: train a classifier and then use it to classify hand movements.

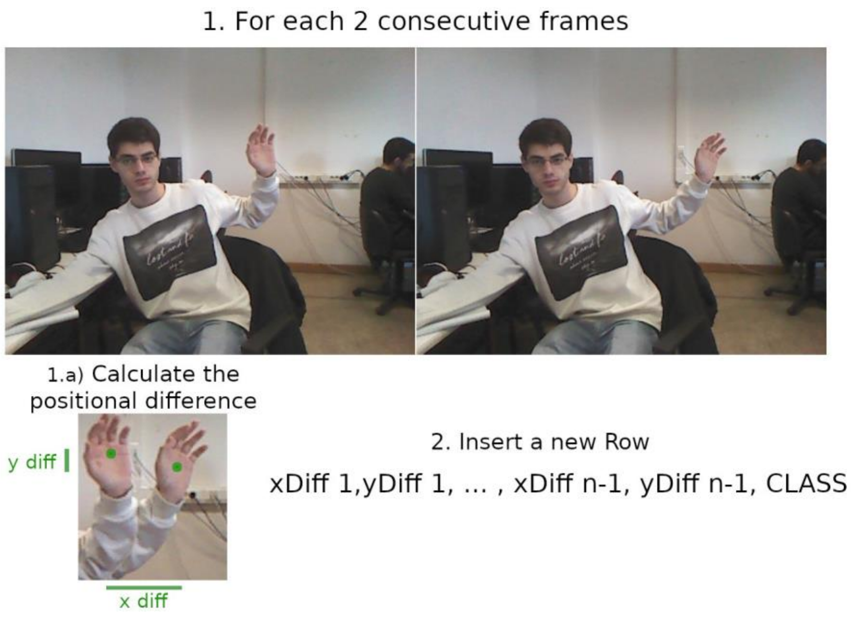
1. For each 2 consecutive frames
1.a) Calculate the positional difference
y diff
x diff
2. Insert a new Row
xDiff 1,yDiff 1, ... , xDiff n-1, yDiff n-1, CLASS

Figure 7.4: Normalization of centroid values

7.1.2 The working solution

The main problem when attempting to classify a hand movement as handwaving is to not classify any other type of hand movement as handwaving. Because of this, the main focus of the brainstorming was to identify what distinguishes a handwave from any other type of hand movement. The main identified characteristics are:

- The hand is facing the person that we are trying to call.
- The hand is open. In other words, it is an open palm.
- The movement of the hand describes a wave, hence the name.

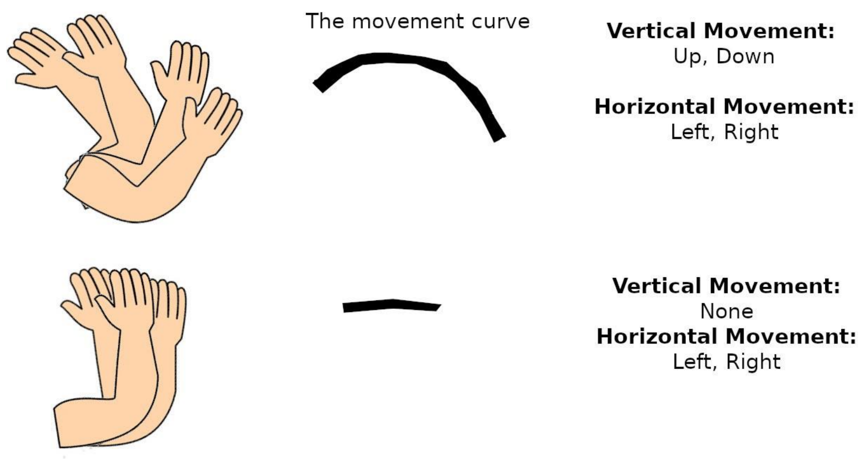

Figure 7.5: Study of the movement of the hand in both axis

The ’wave’ movement can be dissected into two axes: horizontal and vertical. In the horizontal axis, the hand performs a sequence of back and forth movements (left to right and vice-versa). In the vertical axis, the movement of the hand can be more complex depending on the inclination of the forearm and the angle that the arm makes with the trunk.

In the figure 7.5, two different waving cases were exposed. In both cases, the hand moved back and forth in the horizontal axis. However, in the second case, the hand barely moved in the vertical axis. The idea of this approach is to completely ignore the vertical axis. This has to do with the idea that humans don’t do any other type of hand movement with an open palm and moving the hand back and forth except when they are handwaving. Therefore, these two conditions constitute the foundations of the algorithm, as they are what separates handwaving from any other type of hand movement. This way, we achieve the end goal and prevent problems that may arrive by considering the vertical axis, such as the following: in two consecutive frames the hand detector detected the hand in slightly different positions, however, the hand didn’t move in the real world. If we were to consider the vertical axis then, because of the second case represented in the previous image, this difference has to be considered as a movement when in reality there was no movement. Therefore, by disregarding the vertical axis we achieve the same goal and avoid a few problems.

The algorithm gets RGB and Depth frames from the Kinect camera. The depth image returns the distance of each point to the camera in millimetres. For each frame, the hands and faces are obtained using the hand and face detectors. Afterwards, the hands that aren’t palms are filtered out. This is done by extracting the hand from the image, using the box dimensions provided by the hand detector, and then passing it to another detector. This detector is also based on a cascade classifier. It was initially proposed by Paul Viola and improved by Rainer Lienhart. It detects palms or fists in an image. By only passing it the area of the image with the hand we prevent it from detecting palms in another region and wrongly classifying the current hand as a palm. If the detector doesn’t detect any palms in the region then the hand is discarded.

Afterwards, the depth value of each face and hand are obtained. First, the x and y coordinates of the centroids of the hands and faces are obtained. With these coordinates, one could obtain the depth value from the depth image. However, the Kinect camera isn’t 100% accurate. Therefore, by only using only one value we are risking carrying a lot of errors into the next steps. Because of this, a function was created to calculate the median of the depth values around the centroid. An average could have been calculated instead of the median, however, the idea was to filter out the outliers as the problem isn’t when the depth value is inaccurate by a few millimetres but when it is inaccurate by a few hundred. Subsequently, the hand depths are compared to the face depths. The hands that aren’t inside of any threshold, centred around the faces depth values, are discarded. This is done because, when handwaving, the hand is usually parallel to the face, in other words, the face and the hand have around the same depth value.

This step removes hands that are ’too in front’ or ’too behind’ of any face, as neither of these cases could correspond to a frame in the handwaving movement. The credit of this step has to be attributed to Luís Pais [Pai19] as this would have been used in his approach.

Next, the position of the hands and faces in the real world is obtained. This is done by using the x and y coordinates of the centroid in the RGB image and the depth calculated previously. It is worth noting that the z real-world value corresponds to the depth. With these three values and by using Kinect constants that depend on its calibration and are based essentially on trigonometry, one can find out the real world x and y coordinates. Thereupon, the hands are filtered one last time, this time by calculating its distance to each face. If the distance to the nearest face is still above a configurable threshold then the hand is discarded. The idea behind this filter is the following: in a frame, there can be a face and hand from two different people, the face of one of them and the hands of the other are outside the frame. Now, a hand needs to be attached to the body, by an arm. Therefore if a hand is farther away to the face than the value which represents the maximum length of the arm then the program detects that the face does not belong to any face in the image and consequently is discarded.

After all these filters there should either be no hands left or only one. Technically there is a chance that there can still be more than one hand if two people are waving at the robot at the same time. However, not only is this case extremely rare but also means that both people are waving and therefore since the robot will detect that there is handwaving in the frame it will be correct. If there are no hands left then a counter is increased. This counter counts the number of consecutive frames where there were no hands left after the filters were applied. If this counter hits a certain configurable value then all the variables used to detect handwaving are reset. These variables will be explained further on. This counter is necessary because if a hand goes out of view for a certain number of frames then it needs to be considered lost. After all, a different hand may then enter the frame and it shouldn’t be considered the same hand. If there are hands in the frame then this counter is reset (set back to zero).

In the case where there is a hand left, its new position is compared with its previous position. If the difference is smaller than a configurable threshold then it’s ignored. As mentioned before, in consecutive frames the hand detector may return a box in different positions even though the hand didn’t move. This threshold mechanism prevents the algorithm from detecting the going back and forth constantly even though the hand is stationary.
The way the handwaving will be detected is if it respects three conditions:

- The hand changed direction x amount of times. Where x is a configurable parameter.
- The average distance traversed each time it changes direction is superior to a configurable threshold (Amplitude).
- The average time it took for the hand to change direction is inferior to a configurable threshold (Frequency).

After calculating the current position of the hand, and comparing it to its previous position, the algorithm obtains the current direction that the hand is moving to. If the direction changed i.e. it is different than the direction it was moving in the last iteration, then the algorithm updates the hand's direction and increments a counter. This counter counts the number of times the hand changed direction. If this counter hits a certain limit then the algorithm will check if the average distances traversed and the average time stamps are superior to a certain threshold. Each time a direction changes a timestamp is added to a list. This timestamp is the time that passed since the hand started moving in that direction and until it changed. Additionally, there is also a list which holds the distance traversed by the hand in that particular direction. Calculating the average of both of these lists we obtain the average amplitude and frequency of each wave. If the time is too slow then even if the hand did move far enough each time, it did it over a long period of time and therefore it was not a handwave. Similarly, if the hand did move fast enough but did not move 'far' enough then it was not a handwave either. This thought process is exemplified in the following 7.6:

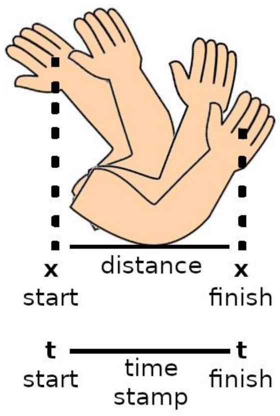

Figure 7.6: How the distance and timestamp of each wave is calculated

The algorithm can be summarized in the following list of steps:

1. For each RGB and depth image captured by the frame:
    (a) Obtain hands and faces in the RGB image.
    (b) Filter out hands that aren't palms.
    (c) Calculate the centroid of each face and hand.
    (d) Calculate the depth of each face and hand.
    (e) Filter out hands that aren't inside any depth threshold centred around faces.
    (f) Calculate the world position of the hands.
    (g) Filter out hands that aren't close enough to any face.
    (h) If there are no hands left, increase a counter and potentially reset variables if it hits a certain value.
    (i) Obtain distance traversed by the hand between two frames (in the x-axis).

(j) Ignore iteration if the distance is below a certain threshold.

(k) Calculate direction (Left or Right).

(l) Update position if the direction didn't change.

(m) If the direction did change:

  i. Add time frame to the list.

  ii. Add distance traversed to the list.

  iii. Increase counter of direction changes if it hits a certain value check if the average of time frames and distances are superior to its respective thresholds and if so, detect handwaving.

7.2 Obtaining the table's position

The robot detects that people are handwaving at it using the handwaving detection module. This module returns the coordinates of the hand. However, these coordinates are relative to the camera, in other words, they are in an axis system whose origin corresponds to the camera. However, the axis system of the camera doesn't match with the axis system of the map, as exemplified by the figure 7.7.

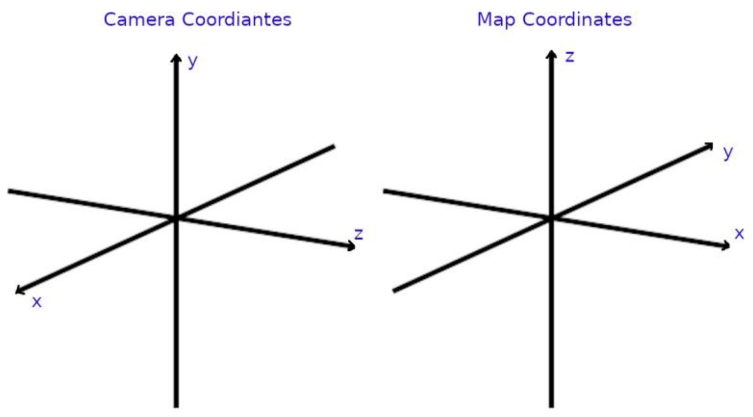

Figure 7.7: Camera and Map coordinate systems comparison

This means that before worrying about anything else, the camera coordinates need to be adjusted as following:

- Map_x = Camera_z
- Map_y = -Camera_x
- Map_z = Camera_y

There is a second problem that needs to be tackled. The coordinate system of the map is centred around the position of where the robot started when the map was generated. However, the coordinate system of the camera corresponds to the robot's current position. Additionally, the robot may have a rotation different from the start rotation of when the map was generated. This problem is exemplified in the following figure.

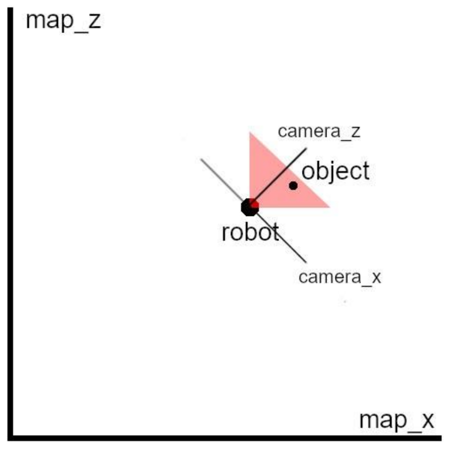

Figure 7.8: Camera and Map vertical axis rotation problem

Because of this problem, rotation and translation transforms need to be applied to the coordinates returned by the detector. Two ways of achieving this were explored: a manual way using trigonometry to calculate the transforms and an automatic way using ROS's TF package.

7.3 Manual Conversion

If one was to add the coordinates returned by the handwaving detector to the robot's current position on the map then the new position would be wrong by a certain amount of degrees. This value corresponds to the robot's current rotation value around the vertical axis. In order to fix this, the point's position on the camera's coordinate system is first rotated around the origin by this value. Only then are this point's coordinates added to the robot's position on the map. Therefore, the resulting formulas for converting the camera's coordinates into map coordinate's are the following:

$$map\_x = robot\_pos\_x + camera\_x * \cos(robot\_rot\_y) - camera\_z * \sin(robot\_rot\_y)$$

$$map\_z = robot\_pos\_z + camera\_z * \cos(robot\_rot\_y) - camera\_x * \sin(robot\_rot\_y)$$

7.4 Conversion using TF

TF is ROS' transform package. It's implemented in such a way that it allows programmers to use the transformation that they require without having to think and implement the math themselves (as it was done above). First, the robot's tree of frames was determined. Each frame is a component of the robot and has its own coordinate system. The frames of the camera and world were identified, they are respectively: `camera` and `odom` frames. A transform listener between these two frames was created. This listener returns the transformations that allow conversion from camera coordinates to map.

The result is divided into two arrays: translation and rotation. Each of these components has four cells as they are quaternions. These two components are then used to obtain a transformation matrix (4x4). Afterwards, the camera coordinates are placed in a 4 wide array (x, y, z and last value is set to 0). Lastly, the transformation matrix is combined with the camera coordinates' array resulting in the map coordinates.

7.5 Getting the order

This section attempts to explain the 4th step of the tasks' algorithm with more detail. At the beginning of the task, the robot will read a text file that contains all the items that are available for order in the menu and load them into its memory. When the robot gets close to the clients' table it will have to note down their orders. It will start by asking the clients what they would like to order. After the clients finish communicating with the robot, it will obtain a list of sentences from the speech recognition module. The robot will search for each item of the menu in each of the spoken sentences. If an item's name is mentioned in a sentence it is added to the order. If the list returned from the speech recognition module is empty, meaning no sentences were recognized, or no item from the menu existed in all the spoken sentences, then the robot will ask the clients: ”will that be all?”. If the clients answer ”yes” then the robot returns to the bar, if not then the robot will continue to accept the orders until the clients respond affirmatively to the previous question.

7.6 Conclusion

This chapter explains the implementation of the restaurant task. The task requires the robot to detect when people call for it by handwaving. Therefore, a handwaving detector had to be implemented. Many approaches were attempted, leading to an implementation based on detecting frequent oscillations in the position of hands close to faces in the horizontal axis. The robot moves between the bar and table locations using the navigation module implemented previously. It obtains the coordinates of the tables by applying a transformation matrix. Lastly, the robot communicates both with the clients and the barman through the speech and speech recognition modules.

Chapter 8

Speech, Object and Person Recognition

This task contains a misname. There isn't a need for person identification since it is assumed that a person is already standing behind the robot. The robot will simply have to rotate and face the person. This was done using the navigation's sub-module rotation to make the robot rotate 180°. Then, the robot will have to get closer to the person. Since it is assumed that there is no other object in between the robot and the human, the move-to-human sub-module that was extracted and improved from Antonio Fernandes' 'help-me-carry' code, was reused for this step. Afterwards, the robot will have to greet the human, this was done using the speech module, the robot will randomly select an item from a list of possible greetings and speak it. For the rest of the task, 2 new modules had to be developed. The robot will have to answer a question correctly so the ability to answer questions had to be created. Additionally, the robot will have to recognize an object that is shown to it and speak its name, therefore an object recognition module had to also be developed.

8.1 Ability to answer questions

From the research that was done, robocup@Home doesn't provide a list of questions that the robot has to answer, or much information about them. Because of this, it is assumed that the competition requires that the robots are capable of answering any type of question. It is not expected that the robot is capable of answering any possible question as that would be an impossible task. Instead, the level of the robot's ability to answer questions is tested. However, the amount of work required to develop software that allows the robot to be competent at answering questions requires its own dedicated project. Therefore, this section will explain what was done as a foundation for potential new projects to expand on and provide the robot with a more competent ability to answer questions.

8.1.1 Implementation

The implementation of this feature can be divided into two components: online and offline. The robot usually has internet access around campus and should have internet access during the competitions. However, there are cases where it doesn't have access to the internet for a variety of reasons. If this is the case, then the robot shouldn't become useless. Because of this, if the robot has internet access it will try to answer the question using its online method, if that is not the case then the robot will resort to its offline method.

The online method is completely dependent on Wikipedia's knowledge. There is a python library, named Wikipedia, that allows programs to search Wikipedia similarly to a browser search. This search returns a list of available page names that match the search query. The search query, in this case, is the question that the user asked the robot. Afterwards, the robot requests the contents of the first page as it's the one with the biggest match. Lastly, the first sentence from the page is returned and will be spoken by the speech module. This approach assumes that the answer is in the first sentence of the page, which is most often the case. However, it is clear to see that there are cases where the first sentence is not what the user asked. Hence, as mentioned before, there is a lot of room for improvement in this specific functionality.

If the robot doesn't currently have internet access or if Wikipedia had no information about the question that was asked, then the robot will try to use its offline method to answer the question. This offline component is what needs to be improved by a dedicated project. It is currently only a placeholder. A proper machine learning method is necessary. Nonetheless, the current approach is composed of having a text file with thousands of questions and their respective answer. These questions were obtained by calling 'opentdb' API which provides a list of questions and answers. Their database keeps expanding as more users create questions so a program was developed to update the current local list with the new questions. These questions are completely random and vary from a wide range of topics. This approach is rudimentary. On one hand, the amount of possible questions that can be asked is much bigger than the few thousands that are available so the question that was asked will most likely not be in the list. While, on the other hand, even if the list was complete, it would be so long that it would be impossible for the robot to hold it in its memory, much less search the question in the list in an acceptable time frame.

In short, the online component is somewhat competent at answering questions but not perfect, while the offline component is very rudimentary. Both work as placeholders and should be replaced if and when a better solution is developed.

8.2 Object recognition

Robocup@Home's rule-book doesn't list all the objects that may be required to be recognized by the robot in the competition. It's very generic in their description such as 'colourful hand-held objects'. However, some objects are mentioned in the provided examples. Therefore, the following list of items was extracted from the objects that were explicitly mentioned in the rule-book. Because there may be objects that weren't explicitly mentioned and therefore are missing from the list, the created dataset is may be incomplete. As a result of this, if new items are listed in newer versions of the rule-book or somehow new information about these items that are required in the competition is discovered, then the dataset should be updated accordingly and the model retrained.

The list of objects that were present in the documentation is the following:

- A plate. 4 different coloured plates were created (red, green, blue and yellow).
- A bowl. 4 different coloured bowls were created (red, green, blue and yellow).
- Cup. 4 different coloured cups were created (red, green, blue and yellow).
- Napkin. 3 different coloured napkins were created (pink, yellow and blue).
- Fork. 4 different coloured forks were created (red, green, blue and yellow).
- Knife. 4 different coloured knives were created (red, green, blue and yellow).
- Spoon. 4 different coloured spoons were created (red, green, blue and yellow).
- Thrash bag.
- Paper bag. 4 different coloured papers were created (red, green, blue and yellow).
- CD/DVD/Blue-ray disk.
- Book. 4 different coloured books were created (black, blue, red and orange)
- Pen. 4 different coloured pens were created (red, green, blue and yellow).

A dataset with these 12 classes, plus an additional class named 'Nothing' that represents when no object is being shown to the robot, was created. Since the task requires that the robot rotates and gets closer to the human, then the point of view from where the objects will be shown to the robot is fixed. Therefore, the images were obtained from this point of view, changing only the rotation of the objects between frames. Google's Teachable Machine was used to generate the dataset. It is a framework that allows for the creation of machine learning models by abstracting all the theory. Only the dataset generation part was used. It provides a simple user interface that allows the users to configure the number of frames to be captured and the time interval between them. In short, it simplifies the process of creating a dataset. 900 images were obtained for each class. Additionally, for items that can have multiple colours such as cups, the 900 images are composed evenly of the colours. This means that for the case of the cup class since there are 4 different coloured cups (red, yellow, green and blue), 225 images were obtained for each colour.

The dataset was split into three subsets: training, test and validation. 10% of the data was used for test and another 10% was used for validation. It is also important to note that the 10% was extracted from each class meaning no class is more or less represented in any of the sub-sets. When it comes to model architecture, 2 types were tested: resnet and inception v3. Only these two were used as inception v3 achieved satisfying results. The results of inception are shown in the table 8.1.

Table 8.1: Coloured Object Dataset Results on an Inception v3 architecture

Epoch	Loss	Accuracy	ValLoss	ValAcc
1	48.03	0.9275	0.34	99.91
2	2.99	0.9946	0.02	100.00
3	2.35	0.9965	0.12	99.91
4	1.19	0.9980	0.01	100.00
5	0.94	0.9989	0.00	100.00
6	0.52	0.9993	0.00	100.00
7	0.40	0.9998	0.00	100.00
8	0.30	0.9998	0.00	100.00
9	0.28	0.9999	0.00	100.00
10	0.28	0.9998	0.00	100.00

The model achieved 100% accuracy on the test data.

The previous dataset is specific to the competition. However, purely red knives or blue spoons aren't seen too often in the outside world. The robot should have utility for university campus usage if no blue spoons appear for it to recognize then its utility is null. Therefore, other datasets were explored to have the robot recognize objects that can be available outside of the robotic competitions. The dataset named CORE50 created by Vincenzo Lomonaco and Davide Maltoni [LM17] was used. It has 10 different classes each with 5 sub-classes of objects that can be handheld. The dataset provides 3300 images for each subclass. Only items that can be handheld are necessary as the intention is for the human to show the object to the robot in their hand. Still, it is not perfect, as ideally, the robot would recognize any type of object that can be handheld. This problem was tackled in the last task.

Once again, Resnet and Inception v3 architectures were tested to determine which one should be used for a model trained with the CORE50 dataset. After 20 epochs the best accuracy the RESNET model achieved in validation was of 70.75% that corresponds to the 4th epoch of training therefore these respective weights were the selected ones. Additionally, this model achieved an accuracy of 87.45% in the test data. After 20 epochs the best accuracy the Inception model achieved in validation was of 75.39% that corresponds to the 3rd epoch of training therefore these respective weights were the selected ones. Additionally, this model achieved an accuracy of 90.05% in the test data. Inception once again achieved better results than Resnet. The results of Inception are shown in the table 8.2.

8.3 Conclusion

This chapter presents an explanation for how the 'Speech, Object and Person Recognition' task was implemented. As mentioned before, it is a small misname as the task doesn't require person recognition. The robot simply assumes that a person is standing behind it and that there are no objects between them. Therefore, it uses the navigation module to rotate 180^{o} and the module extracted from the help-me-carry task to move closer to the human. Then, the human asks the robot a question which it understands through its

Table 8.2: CORE50 Results on a Inception v3 architecture

Epoch	Loss	Accuracy	ValLoss	ValAcc
1	38.16	92.58	1.7267	63.24
2	3.82	99.26	2.4168	56.56
3	2.12	99.59	1.0686	75.39
4	1.50	99.71	2.4497	57.17
5	1.23	99.77	2.9451	52.21
6	1.05	99.81	1.5465	70.17
7	0.51	99.92	3.0387	52.12
8	0.15	99.99	2.4687	63.85
9	0.19	99.97	2.7293	59.06
10	0.23	99.97	2.1623	63.65
11	0.31	99.95	1.9359	67.56
12	0.28	99.95	1.5529	72.10
13	0.34	99.94	2.4394	59.19
14	0.28	99.96	3.2756	51.57
15	0.25	99.97	2.6163	62.03
16	0.14	99.98	1.9606	65.68
17	0.17	99.98	2.8461	58.95
18	0.07	100.00	2.1142	67.18
19	0.10	99.98	3.3059	56.95
20	0.09	99.99	1.6693	72.27

speech recognition module. Two approaches were developed to attempt to answer the question: an online and an offline approach. The first interfaces with Wikipedia's API while the other searches for the question in a text file with thousands of questions. The latter is exceptionally rudimentary and, since the robot should function without internet access, future projects should improve on it. Lastly, in order to recognize the object that the human shows, a classification module was created. Two datasets were used. The first was created by hand, after identifying the specific objects requested by the competition. The second's purpose is to attempt to provide the robot with the ability to recognize objects outside of the competition. Two architectures were tested: Resnet and Inception. Inception outperformed Resnet in both datasets.

Chapter 9

Take the Elevator

In this task, the robot will have to take the elevator to move to another floor of a building. The task starts with the robot already facing the elevator. The robot will have to communicate with humans to succeed in its goal. This help involves selecting the correct floor and detecting that the elevator reached the robot's desired floor. Firstly, the robot has to detect that the elevator's door is open. Then it will have to identify the inside of the elevator to figure out if it can enter without colliding with obstacles, more importantly, humans. After managing to enter the elevator the robot will ask the humans to select its desired floor. The robot waits for confirmation before asking again. It will then wait for people to indicate that the elevator has reached the specified floor. The robot will then exit the elevator. Detailed explanations of how each of these steps were implemented can be found in their respective sub-sections. This task was extracted from SCIROC's Consumer Challenge. However, it provides the robot with an essential capability for campus navigation. For the service robot to achieve its goal, most often than not, it has to navigate its workspace which often has multiple floors. As a result, either the robot can move from floor to floor or its utility is restricted to a single floor. Therefore, by implementing this task not only can the robot perform it in the competition but also use it to move from floor to floor in campus buildings.

Many approaches have been developed and applied to different robots that allow them to take an elevator. However, most of that research is focused on controlling and monitoring the elevator itself [KAO07][TSG+13][ALST16]. This includes recognizing the elevator's control panel to press the correct button for the desired floor, as well as analysing the elevator's floor display to detect when the elevator has reached the robot's desired floor. The process of entering and leaving the elevator, however, is often neglected.

Some choose to ignore the state of the elevator's inside, assume it is empty, and simply enter it [ALST16]. In the current world, modern robots are expected to move autonomously and therefore mustn't pose a threat to not only itself but especially humans that may be inside, entering or leaving the elevator. Other approaches require the robot to have prior knowledge of the elevator's inside [ATC+12][ALST16], that being either a map of the elevator or expecting the elevator to have specific features that help the robot enter it. Service robots should be able to take the elevator without requiring it to have specific features built specifically for it. Lastly, some robots tackle this problem using multiple sensors [TSG+13]. However, if the problem can be solved using fewer sensors, then this ability becomes available to a wider range of robots. Because of this, this paper provides an approach which aims to allow robots to enter and exit elevators safely while limiting the sensors to the bare minimum, that is, one.

9.1 Related work

A state-of-the-art analysis was done to have an idea of how modern robots tackle this problem. In [KAO07], the focus is split on controlling the elevator and navigation. The authors propose a method for the recognition of the elevator's call button for the robot to press, as well as the recognition of the current floor number. By applying an adaptive threshold method to the 2D image captured by the robot they obtain a binary image and create the candidates of the button and the floor number using an additional filtering method. They then reject weak candidates using an artificial neural network and consequently get the position of the button and current floor number. The authors then describe their method for allowing the robot to enter the elevator. They first build an occupancy grid map. Then they calculate the fitness value from the occupancy grid map. The goal position is the one with the maximum fitness value. The path is built using a distance transform.

In [ZCL19] a method is proposed to tackle the multi-floor navigation challenging robots. A solution is provided to the problem of the discontinuity of maps between floors. The authors introduced a 2D maps alignment algorithm using a map server provider that solves the transition between different maps. Afterwards, a method for tracking the state of the elevator's door and the current floor is provided. The detection of the door's state requires the robot to have access to a map of it. A barometer was used to detect what floor the elevator is currently on. Additionally, using a Convolutional Neural Network (CNN)-based object detection algorithm, a method was created to detect and track humans to avoid collisions. Lastly, for path planning, their method was based on the creation of cost maps, Adaptive Monte Carlo Localization (AMCL) for localization and Dijsktra's algorithm as a global planner.

In [ATC^{+}12], the authors developed a method for a robot to navigate using an elevator. The project was done in compliance with Tsukuba Challenge 2011' rules. The method requires the use of a laser range scanner and an omnidirectional camera. The Hough transform was used to detect straight lines in images to detect when the elevator's door opens. When it comes to entering and exiting the elevator the method relies on the two sensors with the dependency that the elevator has a stop line. Template matching and edge detection techniques are applied to omnidirectional camera images to detect the stop line. To avoid collisions with passengers, the laser range finder is used to detect distances to obstacles.

The work presented in [TSG^{+}13] explains the developed method for autonomous multi-floor navigation, using an elevator, applied to a semi-humanoid robot. The robot is capable of mapping and localizing the elevator in its environment, navigating towards it, entering it, pressing the button of the desired floor and exiting the elevator at the correct floor. The robot is equipped with laser range finders, stereo and monocular vision systems, and robotic arms. The work focuses on navigation towards the elevator and the robot's interaction with the elevator's buttons.

The robot enters the elevator in a straight line not considering an ideal position. A SLAM algorithm was used to map the environment, using a LIDAR scanner, and localize the robot in it. The elevator buttons were detected using a technique named FastMatchTemplate. Afterwards, the localization of the buttons for manipulation with the robotic arms is obtained by reading depth information from its cameras.

In [ALST16] a method is proposed for multi-floor navigation for specifically distributed Life Science Laboratories. A solution to the discontinuity between maps of different floors is proposed. Their method proposed a map with multiple layers each representing a floor. An algorithm for localization and path planning was developed for navigation in the laboratories. As for entering the elevator, the authors created a landmark on the maps for the elevators. A method for detecting the elevator's button was developed. The data was captured using a Kinect 1 camera. The images were then processed through a series of filters. After button pressing, the mobile robot continuously checks the door's state and waits to enter the elevator. Then, a landmark on the map is used to help the robot enter and localize itself in the elevator. Lastly, the robot recognizes the destination floor button and controls the elevator similarly to the outside button.

9.2 Pipeline

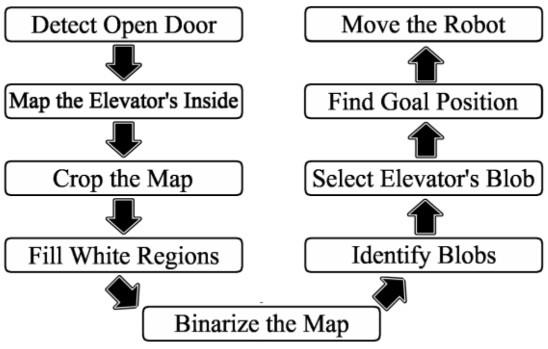

Figure 9.1: Pipeline of our proposal.

The approach assumes that the robot is standing in front of the elevator and facing towards it, as these are the requirements of the task in the competition. Therefore, the robot must know where the elevator is in the building and have the capability of moving to its location. This is done by creating a map of the environment and using the navigation module by specifying the elevator's location on the map. The robot first needs to identify when the elevator's door opens. During this process, it obtains its distance to the door. Once the door opens, the robot creates a map of the elevator's inside. Afterwards, the robot processes the map using the information of its distance to the door. Then, if possible, it figures out a position on the map to move to. Lastly, it moves to that position in the elevator. The robot then communicates with a human to reach the desired floor. The robot first politely asks the humans to select its desired floor, using the speech module.

The robot then waits for confirmation before repeating the request. This is done through the use of the speech recognition module. Afterwards, the robot waits for humans to indicate when the elevator has reached the desired floor. Once this indication has been given the robot politely asks the humans to hold the elevator's door for it. To exit the elevator, the robot rotates 180º and repeats the previous process. Each step of the process is explained with more detail in their respective sub-section.

9.2.1 Detect when door opens

Before the robot can enter the elevator it first needs to know at what distance it is standing from it. This way it knows that if it moves a certain distance forward, it will be inside the elevator. However, it also needs to detect that the elevator's door has opened else it would collide with it.

To achieve these goals, the robot reads information from the depth sensor. In our case, the sensor that was used was a Kinect camera.
Let's consider that the robot is standing facing the elevator. We represent the depth image by *d*, which is a matrix with 640 by 480 pixels, in our case. Denote *D* as the distance from the robot to the elevator. We can estimate *D* by finding the average value of *d*: $\bar{d} = \sum_{i=1}^{480} \sum_{j=1}^{640} d[i,j]/(480 * 640)$.

However, only a portion of the depth image is useful. The frame of the elevator and everything around it may be inside the camera's viewpoint. Because of this, since the objects outside of the elevator are closer to the camera, $\bar{d}$ will be smaller than *D*. Hence, if the robot were to move forward $\bar{d}$ meters it would still be outside of the elevator. In order to solve this, only the center columns of the depth image are kept and used to calculate the average distance, discarding the columns that correspond to the elevator's frame and everything around it: $\bar{d_1} = \sum_{i=1}^{480} \sum_{j=271}^{370} d[i,j]/(480 * 100)$. We chose to use a range of 100 because this amount provides enough precision without risking elements around the elevator to be included.

Additionally, we know that an elevator's door opens horizontally. Since it doesn't open vertically then the vertical information is redundant, because the depth values in cells with the same column are similar. If the elevator's door is closed then all the values will be identical. However, once the door starts to open, the values between columns start to differ. Some have depth values that still correspond to the distance to the door while others have depth values that correspond to the distance to the objects inside the elevator. The latter must have higher depth values because, even if there are objects inside the elevator since they were behind the door, they are necessarily farther away to the robot than the door is. On the other hand, the rows should remain similar to each other, since it is assumed that the door opens horizontally. Therefore, we can simplify the region of interest in the matrix even further and only calculate the average values in a single row: $\bar{d} = \sum_{j=271}^{370} d[120,j]/100$. Only values from the centre row are considered.

For every depth image received from the camera, the robot obtains $\bar{d}_2$. These values are then placed in a queue with a size of S. The smaller the queue the more susceptible the robot is to errors from objects that stand in front of it, while the larger it is, the longer it will take for the robot to detect that the door opened. A value of $S = 50$ ensured that the robot detects that the door opens quickly and is resilient to when people pass in front of it. When a new value for $\bar{d}_2$ is calculated, the robot calculates the difference of this value to the average values in the queue, $\bar{Q} = \sum_{i=1}^{S} Q[i]/S$, before placing it in the queue. When the door opens, the depth value will be the depth value from the robot to the elevator's door plus the elevator's depth. The queue stores depth values from the robot to the elevator's door. Therefore, if the difference between the new value and $\bar{Q}$ is higher than a certain threshold the robot knows that the door opened. This threshold can't be too small else the robot may signal that the door opened when it didn't. Similarly, it can't be too big else if the door does open the elevator may not be deep enough to surpass the threshold value. However, this isn't a problem as the elevator's typical depth does give enough of a margin for error to allow the robot to consistently detect that the door opens using this method.

After detecting that the elevator's door opened, the robot knows the distance to the elevator as it is $\bar{Q}$. Using a queue rather than just storing one value increases the precision of the calculation of the distance as well as the detection of when the door opens. If a person were to pass in between the robot and the elevator during the scanning, then the depth of the closest object would be smaller than usual. When the persons leaves, the depth would increase once again. In this case, the difference would likely be higher than the threshold and the robot would indicate that the door opened when it didn't. Additionally, the calculated distance to the elevator would be wrong. With the queue, we prevent this entirely.

Instead of using depth information to detect when the elevator's door opens, one could have used coloured images and classified the door as open or closed using a machine learning model (for instance, using semantic segmentation). However, the sensor's approach works in all cases while in the classifier's case an elevator's door may be so unique that the model struggles to correctly classify it. Other inconveniences would be the need to acquire many samples of elevator doors to train the method and its high computational cost.

9.2.2 Converting from map units to world units and vice-versa

The need to be able to convert map units to world units as well as the reverse, appeared while developing this task. However, their utility has wider applicability than this task, therefore, they should belong in a module. Because of this, they were added as utility functions to the navigation module. As discussed in the navigation chapter, when a map is saved it generates two files: one is the image of the map and the other is a YAML file that provides information about the map it corresponds to. This YAML file provides the necessary information to do these conversions.

One information the file provides is the resolution of the map. The resolution is the number of meters, in the real world, that each pixel equates to. By default, it is usually 0.05 meters, or in other words, 5 centimetres, this means that 20 pixels equate to a meter. On the other hand, it provides the origin of the map. This is a 2-DIMENSIONAL array that indicates how far in the x and y-axis, respectively, the origin of the map (0,0) is from the bottom left corner of the map. For example, if the YAML file indicates that the origin of the map is [2,1] then that means that if we count 2 meters in the x-axis and 1 meter in the y-axis, starting from the bottom left corner of the map, we should reach its origin. It is worth noting that in reality these values are negative, and it indicates how many meters one has to traverse in both axes to reach the bottom left corner starting in the origin. However, since one wants to find the origin it makes more sense to start from the bottom corner rather than using reverse logic. With these two sets of information, one can convert between the two types of units.

9.2.2.1 Map units to world units

Knowing the row and column of a pixel on the map, one may want to know what is its corresponding coordinate in the real world.

$$origin\_pixel_x = yaml\_origin_x / map\_resolution * -1$$

$$coordinate_x = (pixel_x - origin\_pixel_x) * map\_resolution$$

The first formula finds the pixel coordinate of the origin by using the process mentioned previously. The product by minus one converts the origins' value to a positive one. The second formula converts the distance traversed from the origin to meters. The same conversion logic is applied to the y value.

9.2.2.2 World units to Map units

Knowing the x and y coordinates in the real world, one may want to know what is their corresponding pixel in the map.

$$origin\_pixel_x = yaml\_origin_x / map\_resolution * -1$$

$$pixel_x = origin\_pixel_x + (coordinate_x / map\_resolution)$$

Once again the origin's pixel value is determined. Then the meters are converted into pixel units and shifted based on the origin's value.

9.2.3 Entering the elevator

The next step is for the robot to enter the elevator. A simple approach could be to just move the robot a certain number of meters in a straight line. This number of meters could either be the distance to the door calculated before (plus an additional number so that the robot is fully inside the elevator), or until the distance to the closest object (the back of the elevator) is small enough meaning that the robot is so close to the back that it is inside of the elevator. However, these approaches only work if there are no people to take the elevator with the robot. If this is not the case then in the first approach the robot may collide with people and in the second approach, it may think that a person is the back of the elevator and either stay outside the elevator or in the door's area. Because of this, a mapping of the area needs to be done to know where the robot can move to to get inside of the elevator without colliding with people. The first idea was to map out the elevator's entrance and its inside. Once this map is done the boundaries of the elevator would be specified in the map.

With this information the robot would try to move inside the specified boundaries, rtabmap would update the map preventing the robot from colliding with humans. However, this approach is static. Each time the robot needs to enter a new elevator someone has to map it out first and identify its boundaries. Because of this, a more dynamic approach was attempted. In this new approach, a new map is generated once the robot detects that the elevator's door is open. The robot will then automatically identify the elevator's boundaries on the map. Then it will try to find an area inside those boundaries to where it can move in without colliding with people. Each of these sub-steps is explained with more detail in the following sections.

9.2.4 Obtaining the Map

Recall from the navigation chapter that the current map is managed by a service named map_server. The current map can be obtained by running a command that calls a submodule of map_server, map_saver. One can specify the topic from where the map will be obtained (by default is /map, however, rtabmap uses /rtabmap/proj_map) as well as the name and directory to where the map will be saved to. A new function was added to process_controller module that runs the map_saver command, with the provided parameters, and waits for the map to be saved. An example of a map of the elevator that the robot generates just after detecting that its door has opened is presented in figure 9.2.

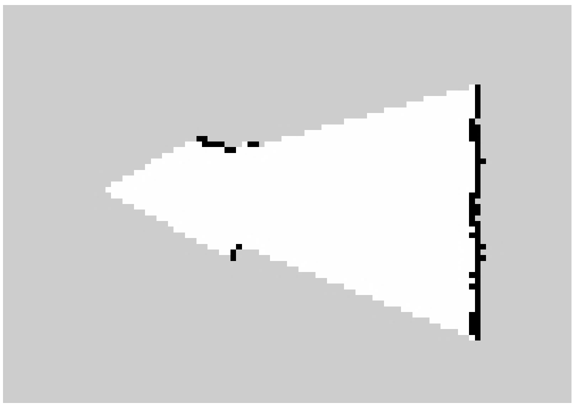

Figure 9.2: Map received from Map Server zoomed in

The robot was able to see a small part of the elevator's door and its back. From here on the robot needs to process the map to identify where it should move to.

9.2.5 Mapping the Elevator's inside

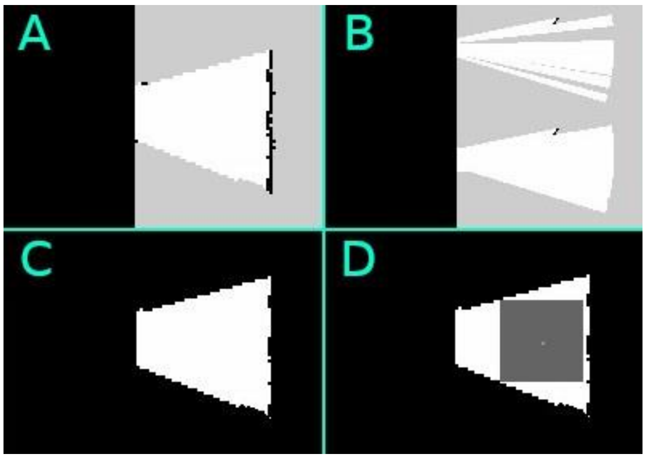

Figure 9.3: Sequence of maps corresponding to the several processing steps of the proposed algorithm. See text for full details.

After detecting that the elevator's door has opened, the robot needs to determine the current state of the elevator's inside. It needs to know if there are obstacles inside it or not to be able to know if it can enter and where it should move to. As a means to achieve this, the robot creates a map of the elevator's inside. An example of such a map is represented in part A of Figure 9.3. The map likely includes more information rather than just the elevator's inside and therefore that extra information will have to be identified and removed in the following steps.

There are many available mapping options in ROS, such as RTABMapping and Hector mapping. We used GMapping, as other options would sometimes fail to map the department's elevator's inside.

Recall that a map in ROS is simply an occupancy matrix. Each cell contains information about the state of a specific location of the environment. The values of the cells can be one out of three possible options: occupied, empty and unknown. An occupied value corresponds to the colour black on the map, it means that there is an obstacle at this location. The empty value corresponds to the colour white and it signals that there is no obstacle in this location and therefore the robot can path through it. Lastly, the unknown value, represented by the colour grey, signals that the robot hasn't obtained any information about this location.

9.2.6 Cropping the Map

After obtaining the map of the elevator, the robot needs to process it to extract the goal position that allows it to safely enter the elevator. Since the robot is standing outside of the elevator some of the outside areas will be mapped. This information needs to be cropped out as the available area outside of the elevator is useless because the goal is for the robot to go inside. When saving the map one obtains the occupancy grid image but also a file that provides information about its corresponding map. This information includes the resolution of the map, in other words, how many meters in the real world does one pixel in the map equates to. Additionally, it also provides information about the starting position of the robot when the map was generated. Since the robot hasn't moved since the generation of the map, this starting position corresponds to the robot's current position.

The x-axis corresponds to the direction the robot is facing initially. The y-axis is perpendicular to the x-axis. Since the robot is standing perpendicularly to the elevator's door, the elevator's door is then parallel to the y-axis of the map. This means that the elevator's door is, for the most part, in a column of pixels. Hence by knowing the distance to the elevator's door, knowing the robots initial position and knowing how to convert world coordinates to pixel coordinates we calculate the door's column pixel on the map. Lastly, we can black out all the columns before (and including) the door's column as they correspond to areas outside of the elevator. This step is represented in part A of Figure 9.3.

9.2.7 Fill the Map

Sometimes after scanning the area, the map that is generated is incomplete. This behaviour is exemplified in part B of Figure 9.3. In this case, the robot should be able to move because there were no obstacles in that area. However, the mapper didn't generate enough available space for the robot to move to. When the robot fails to move in or out of the elevator it resets and starts the process again. However, the mapper would return the same map and hence fail once again and lead to a never-ending loop. The reason the mapper returns a map similar to the one shown before is believed to be because it didn't detect a wall (or obstacles) hence it is not convinced that there is free space in that area. The map presented in part B of Figure 9.3 was generated after scanning inside a room (while standing outside).

The room is much larger than a normal elevator and no wall was in range. This was done while developing this method and helped to debug problems. However, this likely means that, for the case of an elevator, this type of map has a lower chance of being generated, since elevators tend to be small enough to prevent this behaviour from happening. Nonetheless, a way around this fact was implemented to use in case a map like this is generated.

The implemented solution works by filling the grey pixels inside the blob. Pixels inside the bounding box of the blob are searched one by one. If the pixel is grey then the algorithm will check if there are white pixels to its left and right inside a maximum distance or if there are white pixels above it and below it, once again inside a maximum distance. If either case occurs then the pixel is coloured white. The search distance needs to be small else grey pixels in between two different areas are filled. A value of ten appeared to be enough to fix this problem when it occurred. This approach assumes that there aren't any obstacles in these grey spots else the mapper would have identified them, hence these grey pixels can be assumed to be free spots.

It is always risky to colour grey pixels white and assume that the area on the map is free without confirmation. However, the only grey pixels that are being coloured white are the ones surrounded by true white pixels. We assume that if there were obstacles in the unknown area, the sensor would have detected them and coloured the pixels black. Additionally, this is an optional step as the type of map that requires it was never generated in an elevator environment.

9.2.8 Binarize the Map

As mentioned previously the map has three types of values. For the robot to navigate safely, it needs to avoid areas marked on the map as occupied, represented by the colour black. On the other hand, it is free to path through white regions as the sensor didn't detect obstacles in them. However, when it comes to regions marked as unknown, represented by the grey colour, things get more ambiguous. The area is either empty or occupied but since the sensors didn't collect information about it the robot doesn't know if it can move there safely. However, the worst-case scenario is if the region is occupied and the robot chooses to path there. In this case, the robot will collide with the obstacle and worst of all, if it is a person, then the robot is a threat. Because of this, regions marked as unknown on the map will be considered obstacles by the robot. If it doesn't get confirmation that the area is free it will not move through it. In the end, this means colouring the grey pixels of the map, black. This process is represented in part C of Figure 9.3.

9.2.9 Find goal position

The last step of map processing is to identify the 'ideal' position for the robot to achieve in the elevator. The bounding box of the white region is searched pixel by pixel from left to right and from top to bottom. For each iteration, the biggest square composed only of white pixels and with a bottom left corner corresponding to the current pixel is identified. If the biggest square found in the current column of pixels is smaller than the area of the map occupied by the robot, the algorithm finishes earlier. This is because since the blob is searched from left to right if no square bigger than the robot's size at a specific column exists then the robot can traverse past it, therefore, there is no need to continue searching. The centre of the biggest square obtained is the ideal position and it is where the robot should aim to move to. This step is exemplified in part D of Figure 9.3 where the biggest square identified is represented in dark grey and its centre pixel is the goal position. If during the processing of the map no blobs are identified or no square big enough is available, then the robot will remap the area and try again.

9.2.10 Move inside

To perform this step the navigation module will be used. We have created the map. Then, we have localized the robot on the map, since the robot hasn't moved its location is zero in all axis. Therefore, all that is left to do is place a navigation goal to the desired location. This location in world coordinates is obtained from converting the map coordinates obtained from the previous step. The navigation module will create a path and move the robot to the ideal location.

9.2.11 Exiting the elevator

The robot entered the elevator and hasn't moved since. It is important to point out that when the navigation goal was set to enter the elevator, the orientation of the goal position was set to the initial orientation. This means that when the robot enters the elevator it is facing the same direction as it was at the start of the method. Because of this, the robot is facing opposite the elevator's door. Therefore the robot is first rotated 180º, using the rotation function developed in the navigation module, causing the robot to face the door once again, only this time, from the elevator's inside.

To exit the elevator, the robot repeats the whole process used to enter. This means that the robot will have to remap the area, only this time the area corresponds to the environment outside of the elevator. Then, the map is processed through the same pipeline obtaining a goal position. Lastly, a navigation goal is set for that goal position causing the robot to exit the elevator. One thing that hasn't been explained is how the robot knows its current distance to the elevator's door to process the map (cropping step). The distance to the door can be obtained by subtracting the distance that the robot moved in the x-axis by the distance to the door that robot calculated when it entered the elevator.

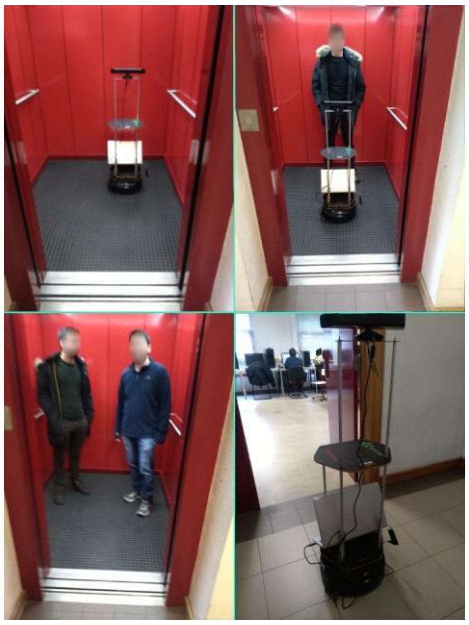
Figure 9.4: Robot entering an empty elevator; Robot entering an elevator with one passenger; Robot entering an elevator with two passengers; Robot entering a room.

9.3 Experiments

Several experiments were done to test our method, these are represented in Figure 9.4. The robot is a turtlebot 2 controlled by a laptop with 16GB RAM, i7 1.8GHz 8 core CPU and 131 GB Disk. First, the robot was tested in elevator environments. Because the elevators are public and people needed to use them, we had time constraints. Because of this, only three different elevators were used for testing. Additionally, since we believe our method can be extended for room switching, the robot was also tested in a room environment. Every experiment was repeated 3 times. In the first scenario, there are no people inside the elevator (or the room), in the second scenario there is one person inside and in the third, there are two people. In all scenarios, there is enough room for the robot to move. All the data captured by the camera was recorded using *rosbags* and is provided for future bench-marking.

The robot was placed in front of the door perpendicularly facing it. The robot was then switched on, followed by the initialization of the recording of *rosbags* and lastly, the program was started.

9.3.1 Detect the Opening of the Door

The first step of the experiments involved testing the method's accuracy at detecting when the door opens. If the door was initially closed the robot detected when it opened with 100% accuracy. Additionally, the method to obtain the distance to the door also showed to be precise as the robot never moved to areas outside the elevator. Even in the room environment, where the door rotates rather than slides, the method worked well. However, the method failed if the door was open right from the start.

This is because the method first needs to obtain the distance to the door and then detect when this distance changes. If the door is already open the distance will only shorten when it is closed. However, once the door closes the method starts working because the wrong distance values in the queue will be replaced with correct ones. Even though it is a setback, it doesn't negate the robot's success, but it does delay it. Since this part of the process can be extracted and tested separately, another independent experiment was conducted. The door was opened, then closed 20 times. The method accurately detected when the door opened and closed in all cases.

9.3.2 Entering and Exiting the Elevator

Afterwards, the pipeline was evaluated in its entirety. The experiment was repeated twelve times with a perfect success rate even in the cases where there were two people inside the elevator. However, one problem was detected. The robot moved to a position closer to the door and farther away from the back of the elevator to avoid the people. Because of this, the robot ended up in a position where part of itself would still be in collision range with the door if it were too close, preventing the elevator's door from closing and consequently preventing the elevator from moving. This is because the map is cropped at the door's position, but the door has a certain depth. Hence, a change to our algorithm had to be done. Before cropping, the robot first adds 50cm to the value of its distance to the door. Even though the elevator door's vary, their depths tend to be similar, and the robot won't fail to enter the elevator if it added a few unnecessary extra centimetres to the door's depth. With this change, the robot crops the map at the point where the elevator's inside starts and this prevented the previously mentioned case from occurring.

To exit the elevator, the method worked similarly to what happened when entering it. If there was an obstacle close to the elevator in the outside, the robot would place itself in a position where it would intersect with the elevator's door. Therefore the change that was done in the previous step was also applied in this step.

9.3.3 Results

The robot succeeded in entering and exiting the elevator as well as the room in all twelve experiments. However, some problems did occur while performing the experiments which forced the experiment to be restarted. A problem that occurred had to do with image compression while saving the map. These problems were related to the processes responsible for gathering information from the sensor as well as mapping. Because these problems are external to the method, the experiments were not considered failed, instead, they were restarted. The processing of the map when entering and exiting the elevator took, on average, 3.2 seconds. The experiments averaged an execution time of 96.92 seconds. These results are presented in Table 9.1.

Table 9.1: Time of execution of the twelve experiments, in seconds.

Experiment	Map Processing Entering	Map Processing Exiting	Total Time
1	2.91	2.84	104.26
2	3.02	2.56	107.43
3	3.04	2.73	100.61
4	3.28	3.48	96.80
5	3.13	3.10	90.28
6	3.24	2.18	104.70
7	2.94	2.68	93.59
8	3.06	2.76	93.49
9	3.18	2.82	91.23
10	3.56	3.60	85.66
11	3.41	5.14	100.05
12	3.58	4.96	94.95
Average	3.20	3.24	96.92

The long average time of the experiments is a consequence of a few diverse factors. At the start of the experiments the door is closed, and it remains this way for a few seconds. This time hovered between 10 to 15 seconds. This way we make sure that the robot is actually properly detecting the opening of the door and not somehow accurately guessing if the door were to always open at the same time. Then the processes of mapping the elevator's inside needs to be launched. Next, the information from the sensor needs to be captured to generate the map. Afterwards, the map needs to be obtained from the mapping process. Then, after applying our map processing pipeline the processes responsible for controlling the robot need to be launched. Finally, the robot needs to move the calculated distance. Each of these steps occurs twice, when the robot wants to enter the elevator and when the robot wants to exit it.

9.4 Conclusion

This chapter explains the implementation of the elevator task. The implementation of this task not only allows the robot to perform the task in a competition but it also allows it to navigate in between floors. The goal was to create a method that worked on novel elevators. In other words, a method that works on any elevator without requiring a specific setup. The implementation avoids collision both with the elevator's door and with obstacles that may be inside it. This is done by detecting when the elevator's door opens and dynamically mapping the interest region (either elevator's inside or outside). Experiments were performed to evaluate the method. Results proved that the method never failed to reach the goals of moving the robot in and out of the elevator. However, time measurements proved the method to be slow due to the time it takes to launch ROS processes, mapping and saving the map.

Chapter 10

Welcoming and Catering Visitors

This chapter explains the implementation of two tasks: 'Welcoming Visitors' and 'Catering for Granny Annie'. These two tasks were combined into one chapter for a few reasons. First, both tasks were extracted from the same competition. Secondly, both tasks are relatively small in scale, especially when compared to the other tasks implemented in this thesis. Lastly, the two tasks share a common goal, which is to identify an input to determine what is the respective task that the robot must do. The 'Welcoming Visitors' task expects the robot to identify people present in images it receives from the camera. Then, depending on the identified person, it performs a task. For example, if the identified person is the mailman then it would probably expect to be handled letters. Similarly, the 'Catering for Granny Annie' task requires the robot to recognize verbal commands and identify what it must do to fulfil the command.

10.1 Welcoming Visitors

In this task, the robot has access to camera images. It needs to try to identify the person that is present in the images. Then, depending on who the person is and assuming the person was identified, the robot performs a specific task. In a nutshell, this task challenges the robot's ability to identify people. The task is specifically used in the competition to identify people that are at the entrance door of a home, through a surveillance camera. Before explaining how this feature was implemented credit needs to be attributed to Vasco Lopes and Nuno Pereira. They had previously developed a program that uses a python library named face_recognition. They provided me with a sample program and hence spared me the time and effort of looking for solutions to this problem as well as figuring out how the library works.

10.1.1 Person Identification Algorithm

The robot keeps a folder of pictures of people it knows. Even though one picture per person could work, as it will be proven during the bench-marking of this algorithm, as the number of known people increases the number of samples per person will have to increase for the algorithm to remain competent. Therefore, a folder of pictures per person will be saved in the database folder. This second approach does, however, require more memory to store all the photos and more processing time to process them all. Initially, the database folder is empty because the robot doesn't know anyone. The algorithm starts by processing the class folders. A list of names is obtained as well as a list of respective image encoders.

These encoders are obtained by passing the images through a function presented in the face_recognition. In a nutshell, the robot has an image database of people's faces and when this algorithm starts it loads it into memory.

Afterwards, the robot starts to obtain images from the camera. The face detector module is used to check if the frame has a face of a person in it. If not, then the image is ignored, otherwise, the image is placed in a queue. This queue stores a set of recent images that have faces in them. This queue is reset if the time between frame insertions is above a certain threshold. This way if one person comes in front of the camera, leaves and then a different person arrives, the frames of the first person are deleted. This queue also has a maximum size to avoid old frames being processed as well as avoiding long processing times. When the queue is full, then the set of frames is processed. This way the robot has more information about the person rather than just one frame. This means that it has a higher chance of identifying correctly a person that it should identify. The robot fails this task if it asks the human to identify a certain person, because it believes them to be a stranger, only to be told that it should know who the person is and therefore it is failing its job.

To identify the person in the images, the robot will use the database. For each frame, it will obtain its encoders. Then it will compare these to all the encoders in the encoder's list. This last step returns a list of boolean values with the same size as the database's encoder's list. Each item in this list indicates if the encoders in the same index from the database's list matches with the frame's encoders. If all values of the list are false and if the robot would only use this single frame, it would guess that the person is a stranger because the encoders don't match with any of the known ones. The class with the most positive indexes (true values) is the predicted class. After repeating this process for all frames in the list the robot obtains a list of results. Some of the results may indicate a stranger while others may guess a different person. A conclusion needs to be taken from these results.

First, the robot will check if the results classify the person as a stranger. It will count the number of results that predict a stranger. Then it will divide this value by the size of the queue. In other words, it will obtain the percentage of stranger guesses. Then it will compare this percentage with a threshold. If it is above this threshold it will end the iteration and predict stranger. If this threshold is too small then the robot has a higher probability of classifying known people as strangers but it also has a lower probability of classifying a person as someone else. For security purposes, it is preferable for the robot to call someone known a stranger than opening the door to one.

If the person is not a stranger then the robot will count the percentage of each name that was guessed, similarly to the stranger percentage. The person with the highest percentage is then returned.

It is worth pointing out that this algorithm is implemented to recognize a person in the frame. It trusts that as long as there is one known person in the frame, the door can be opened. If the algorithm were to require potentially more people to be at the door at a time and all to be identified it would need to be changed.

Lastly, to estimate how well the face_recognition algorithm performs, it was bench-marked on a dataset. The requirements for the dataset were to contain only images of people's faces, have a significant variety of people and have a significant number of samples per person. After searching through different datasets the chosen one was the Faces94 dataset [Uni94]. This dataset contains pictures of faces of 121 people as well as 20 pictures taken per person, varying in facial expression.

A program was developed whose goal is to split the dataset into a test and a training set. The training set will simulate the database while the test set will simulate the images obtained from the camera during the execution of the algorithm. The program receives as input the number of images per class (person) that are to be placed in the training set. Since there are 20 images per class in the original dataset, this number must be inferior to it to ensure that there is at least one image (per class) in the test set. The program first creates the test and training folders. Then it proceeds to create the class folders inside both test and training folders. Next, it iterates over the class folders of the original dataset, selecting $\overline{N}$ random images and placing them in the respective class folder in the training set, while the remaining 20-$\overline{N}$ images are placed in the test set. This program allows us to test on varying test and training set sizes by simply changing the input parameter.

The bench-marking program first reads the training set folder, extracts the descriptors from each image and creates the database. This previous step is the same as the database loading step. Next, the program reads the test folder. For each image in the test folder, the program first extracts its descriptors. Then, exactly as what was done in the developed algorithm, the programs matches the descriptors with the database, obtaining a boolean array then extracts the predicted class from the array by finding the class with the most positive values. Lastly, the program compares the predicted class of the image with the correct class. In case of a match, a variable that keeps track of the number of correct class predictions is increased by one. After iterating through all the images the algorithm's estimated accuracy is obtained by dividing the previous variable by the total number of images in the test set.

This evaluation was done multiple times on test and train sets of different sizes. The results obtained are presented in the table 10.1.

Table 10.1: Accuracy of face identification algorithm on Faces94 dataset with varying number of classes, training set sizes and test set sizes

# Classes	Training/Test Set Size	Total# Test Images	# Correct Predictions	Accuracy %
10	19/1	10	10	100.00
10	10/10	100	100	100.00
10	5/15	150	150	100.00
10	1/19	190	183	96.32
25	19/1	25	22	88.00
25	10/10	250	220	88.00
25	5/15	375	329	87.33
25	1/19	475	358	75.37
50	19/1	50	44	88.00
50	10/10	500	439	87.80
50	5/15	750	647	86.27
50	1/19	950	718	75.58
151	19/1	151	106	70.20
151	10/10	1510	1039	68.81
151	5/15	2265	1443	63.71
151	1/19	2869	1219	42.49

From the results presented in table 10.1 WE can conclude that as the algorithm identifies more people (the number of classes increases), it increasingly struggles to accurately identify the person. However, the results also show that the previous problem can be mitigated by increasing the database size.

10.2 Catering for Granny Annie

In this task, the robot must recognize verbal commands. Then, for each command, there would be a respective task that the robot would perform. However, the linking between command and task can be infinite and easily changed. Therefore, this implementation focuses only on the identification of which command was spoken and the return of variables which would affect the performance of the respective task. The list of commands is the following:

1. Turn the lights [ON/OFF] in the [ROOM] - the robot has remote access to a room's light controls. It can then turn the lights on or off.
2. Leave the blinds [OPEN/CLOSED/HALF-OPEN] - the robot has remote access to the blinds controls. It can then lower or raise them.
3. Fetch the [OBJECT_NAME] - the robot is requested to fetch an object. The robot must understand which object was requested. It will then have to find it, grab it and deliver it. The grabbing part will have to be ignored for the time being as the robot doesn't have an arm yet.

These commands were the ones required by the competition at the date of the development of this thesis. This means that the list of commands may vary in the future and therefore the command identification algorithm will need to be expanded accordingly. However, the implementation of these commands will hopefully provide a good foundation which eases the addition of new commands. The commands in the list contained variables that affect what the robot would do. This means that not only does the robot need to identify what type of task it must perform, but it also needs to know how it must perform it. For example, the first command requires the robot to manipulate the lights. However, the robot needs to know if the lights need to be turned on or off, as well as to which room does this task need to be performed on.

10.2.1 Algorithm

The robot will constantly be performing speech recognition. After a sentence is recognized it will pass it to this tasks' function. This function will return an index number which corresponds to a commands index as well as the value of the variables accepted by the command. If the returned index is negative, then no valid command was recognized.

The function verifies if the input string corresponds to each valid command. Each command has an automata created from the command's grammar specified in the previous list. An automata is a system which represents a grammar, that receives a string of characters and indicates if it belongs to the grammar or not. The automata is constituted by a set of states. The characters of the string are read one by one, changing the state of the automata. If by the end of the string the automata is in an acceptance state, then the string belongs to the grammar, otherwise, it does not. If the input string is accepted by the automata, then this means that human spoke this command. The respective index is returned as well as the values of the variables. The values are extracted from the string using the automata. A representation of the automatas is presented in figures 10.1, 10.2 and 10.3. The asterisk symbol corresponds to a path that accepts any character that doesn't have another valid path from the current state. Furthermore, the automatas change state after processing every single character. However, to simplify the explanation, the automatas inn the figures accept a sequence of characters in between two states, just remember that the two stages are representing multiple.

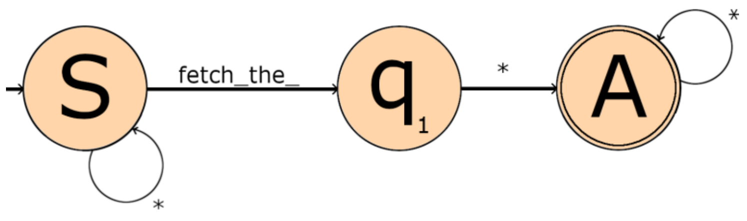

Figure 10.1: Automata for command to fetch objects

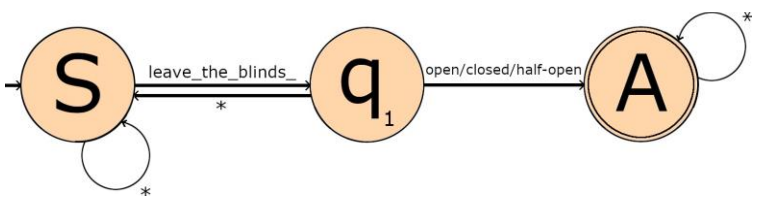

Figure 10.2: Automata for command to control the blinds

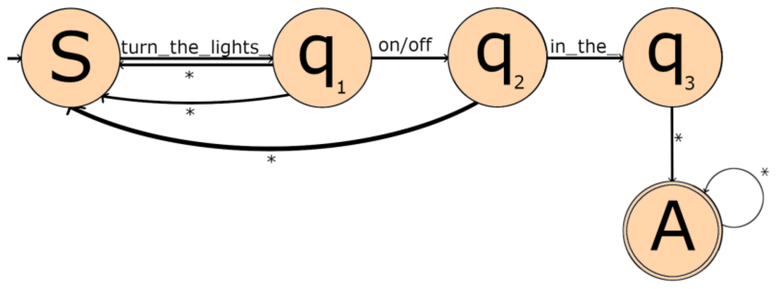

Figure 10.3: Automata for command to control the lights

10.3 Conclusion

This chapter explains how both 'Welcoming visitors' and 'Catering for Granny Annie' tasks were implemented. The first requires the robot to monitor camera images and detect when people are present in the frames. Then, it must attempt to recognize them or classify them as strangers. The algorithm extracts descriptors using the face_recognition library. It then combines information from multiple and sequential frames to achieve the conclusion. The quality of the descriptors was benchmarked on faces94 dataset. The results were satisfying. However, since the algorithm doesn't make a decision based on a single frame, the accuracy of classification is greater than the one reported. The second task requires the robot to understand three verbal commands. An algorithm was implemented using the speech recognition module. This algorithm receives the recognized speech and, based on the results of the different automatas, returns an index, which indicates what type of command was identified, and the values of the command's parameters. Both tasks act as a setup for something else since after recognizing the person or the command the robot could perform any type of task.

Chapter 11

Getting to know my home

This task focuses on acquiring and representing knowledge about the environment. Some random changes are then made to the environment. The robot must identify these changes and point them out. The task 'Getting to know my home' therefore consists of two parts. First, the robot must obtain information about the environment where the task will be performed. The information must be saved in a semantic map which contains the names of the objects and their location in the environment. The process of obtaining this information isn't restrictive since the competition doesn't evaluate this step it simply considers the semantic map as a requirement for its successor. This means that a human could create the semantic map by hand, writing down the objects' names and calculating their locations. However, as will be discussed shortly, a more automated approach was implemented. The second part consists of the robot traversing the environment and verifying if the objects registered on the semantic map are still in their initial location, verbally communicating each conclusion.

11.1 Acquiring information

This step precedes the actual task. There are no limitations as to how the robot should learn new objects. This means that participants could attempt to build classification models during the set-up days. However, each time new objects are added or a new environment is used, the models need to be retrained. Not only would it be challenging to create a competent model in time but it would likely have to be discarded and recreated later. Because of this, a new approach is required, preferably, an approach that eases the processing of teaching the robot about each object.

With the previous goal in mind, an approach was created where the robot follows the human around and learns one object at a time. The algorithm consists of the following cycle:

1. The robot follows the human around.
2. The human tells the robot to stop following.
3. The human points to an object and speaks its name.
4. The robot captures information about the object and saves it in a database.
5. The robot registers the object's position in the semantic map.

11.1.1 Following the human

This step is similar to what was done for the ”Help-me-carry” task. The robot will follow the human around using the follower process. However, the robot will be told to stop when it is time for it to learn a new object. This means that the robot will have to stop the follower process. However, by doing this it also stops the publication of camera images which are essential for learning the new object. Additionally, after learning the new object, the robot will have to register its position on the map, therefore it must know its current location. Because of these two requirements, both AMCL (for localization) and follower processes have to be running at the same time. However, they are incompatible because, since they both require the camera, they share nodes. Because of this problem, a custom launch file that starts AMCL and the setup nodes for the follower process was created. Then, a second launch file launches the follower process, but only the follower node, as it expects the setup nodes to be already registered. The task begins by starting this process, meaning that the robot will be able to locate itself on the map through AMCL. Additionally, a subscriber to the image topic is created which saves the latest image in a variable, while a second subscribes to the acml_pose topic which constantly updates the robot’s position. Next, the process that starts the custom follower process (compatible with AMCL) is started, causing the robot to follow the human around. The robot will use the speech recognition module and constantly detect speech. Once the detected speech matches a known stopping word such as ”stop”, the robot will stop the following process and therefore stay in place. The human will then teach the robot the new object, a process which will be discussed in the following sections. Afterwards, the robot will ask the human if the task is finished. In the case of a negative answer, the robot will resume following the human by restarting the follower process, learning new objects until the human’s answer is affirmative.

11.1.2 Detect when the object’s name is spoken

The human must name the object to the robot because the robot will need to, later on, point out the objects that are present or missing in the environment using their names. The robot expects the human to say the object’s name using the following syntax:

- This is [article] [object_name].

The possible articles in the English language are ’a’, ’an’ and ’the’. Because the speech recognition software isn’t perfect, to minimize delays by requiring the human to repeat the sentence, if no article is recognized, the sentence is still accepted. The algorithm for this step uses the speech recognition module to recognize all the speech being spoken to the robot. The recognized text is concatenated and saved. When the sequence of letters ’this is’ is recognized then the robot knows that the following text equates to the object’s name. If articles are present in the robot’s name, they are removed.

11.1.3 Obtaining information about the object

The robot receives RGB images and point clouds from the camera. Similarly to what was done for the handwaving detection module, a synchronizer was used to guarantee that the RGB images correspond with the point clouds. The amount of data to save per object is one by default but can be changed to save more information each time. Since the robot is standing still at this point, the images and point clouds will be similar to each other, hence the robot would likely just be saving redundant information if the amount was superior to one. However, a bigger amount of data to collect reduces the risk of having to start the learning process of this object from scratch. An example of this is the pointing finger position detection step. The algorithm may not detect the position of the pointing finger in an RGB image and therefore require a new image. With the purpose of efficiency and time saving, the synchronization and saving of RGB images and point clouds are started when the task starts. The images and point clouds are saved in their respective queues. These queues have a maximum size of 'N'. If the queue is full then the oldest value in the queue is deleted and replaced with the most recent one. This prevents outdated data from occupying the computer's memory. When the human speaks the object's name the 'N' most recent RGB images and point clouds are retrieved from the queues. This way no time is wasted starting the camera processes, initializing the subscribers and waiting for N images and point clouds.

The RGB images include the object of interest, the hand of the human pointing at the object and potential background objects. It is necessary to automatically know what object the human is pointing to. A hand key-point extraction algorithm was found and adapted. This original algorithm received an image and, if a hand was present, it identified the pixel coordinates of key-points of the hand. These key-points are returned in an array of 2D-values (x and y coordinate). This array always had a fixed size of N, a configurable number of key-points, by default 22. Some elements of the array could be 'None' values if the key-point was not identified in the image. This is important because after some testing it was also found that the pointing finger's position always corresponded to the eighth point in the array, for the default N value of 22. Therefore, the original algorithm was changed to only return the eighth value in the array, obtaining the position of the pointing finger in an image.

For each image received, the position of the pointing finger is calculated using this algorithm. If the key-point was not detected, it is a 'None' value, the image and respective point cloud are discarded.

When it comes to extracting the object being pointed at from the information received from the camera a semantic segmentation model on the RGB images was used.
While image classification simply attributes a class to the whole image, semantic segmentation attributes a class to each pixel of the image. It is essentially classification per pixel, rather than per image. Although, to classify a pixel it requires information about its neighbouring pixels.

Each colour corresponds to a class. The result is that the individual objects are identified, as well as their positions and contours in the image. By applying a segmentation model to the RGB images received from the camera, ideally, all the objects would be segmented. Then, by knowing the position of the pointing finger, the object at that position would be kept while all the others would be discarded. However, as with any machine learning method, semantic segmentation requires a dataset. Ideally, since any object could technically be shown to the robot for learning, the dataset would be complete with all possible objects in the world. However, not only does such a dataset not exist but would also take too many resources to train a model for it. For that reason, datasets with home objects were explored since that is the type of environment primarily focused on the task. With this said, we seem to have hit a paradox, we want the robot to learn about a new object, however, require the object to be known by the robot when the segmentation model was trained. This isn't exactly true. Yes, the semantic segmentation model requires a dataset and will be used to learn about the object being shown. However, if the object being shown isn't in the dataset, the model will still output a semantic segmentation of the input image and, hopefully, that output can be useful for learning about the object.

Two datasets were found and a model trained for each of them was used and tested. The first dataset is COCO [LMB+14]. COCO is a large-scale object detection, segmentation dataset with over 300 thousand images of everyday scenes and 91 object categories labelled over 2.5 million instances. The second dataset used was ADE20K [ZZP+17] developed by researchers at MIT. It contains 20 thousand images annotated with 150 object categories.

After passing the RGB image through the semantic segmentation model a segmented image is obtained as output. This image has potentially multiple regions, each corresponding to what the model thought was an individual object. We still want to extract the object being pointed to and discard the rest of the scene. From before, we've obtained the pixel coordinates of the pointing finger. Hence the object should be in the segmented region that contains this pixel. The individual regions are searched one by one until the one that contains this pixel is identified. Semantic segmentation attributes a single class to each pixel and does this to all pixels. Therefore this pointing finger pixel can only be in a single region and there is always one region that contains it. After identifying the matching region all the pixels that don't belong to it are coloured black in the original image while the ones that do belong are left untouched.

At this point, depending on the quality of the segmentation, we have an image with the object of interest and everything else blacked out. The robot now needs to extract information about the object in the image to recognize the object when new images are obtained. Image descriptors were created for this purpose. They extract specific features from the image that, hence the name, describe the image. Similar images have similar features while different images have different features hence their descriptors should be very distinct. The idea is to save these descriptors in a database.

Then, when a new image is obtained, its descriptors are extracted and matched with the ones in the database to know which object is in the image. Three different feature extraction algorithms were used: Speeded Up Robust Features (SURF), Scale-Invariant Feature Transform (SIFT) and Oriented FAST and rotated BRIEF (ORB). All of these algorithms are implemented in opencv library. These are hardcoded feature extractors. They were created back when convolutional neural networks didn't exist. Simple tests proved them to be rudimentary for this purpose hence they can't be used.

For processing the point clouds the goal remains the same, that is, extracting the object from the scene. Point clouds provide an extra dimension to the problem, the depth. While in the case of images the scene is projected to a single plane and, therefore it is harder to separate adjacent points. However, in the case of the point cloud these adjacent points may have high contrast in their depth values, this likely means that they belong to two different objects.

An Euclidean clustering algorithm was implemented. Clustering means organizing similar data values into groups where each group is a cluster. The algorithm receives a point cloud and filters it. First points with 'NaN' values are discarded. Then, a voxel grid is applied to the point cloud. Voxel grid creates a three-dimensional grid, where each cell shares the same size N. All the points inside a cell are transformed into a single point. This transformation can vary, but it can be for example the average of the points in the cell. The bigger the N the more values will be in each cell. This consequently reduces the size of the original point cloud.

Then a Kd-tree representation of the point cloud is built. This algorithm will return a tree object that represents the original point cloud. The way the algorithm works is it selects a dimension out of the set of dimensions. Then it calculates the median of the values in that dimension. Points whose value are inferior to that dimension belong to a different branch than the ones that are superior to the median. The process is then repeated for all the dimensions.

Then an empty list C is created. This list will contain the clusters. Additionally, an auxiliary queue of points Q that need to be checked is created. The points in the point cloud are searched on by one. For every point searched, the point is added to the queue Q and this queue is also searched. For every point searched in the queue Q, the neighbour points are searched in a sphere of a specific radius. If the point is not in the queue Q then it is added to it. After searching the queue Q this Q is added to the list of clusters as a new cluster and it is reset for the next iteration.

At this point, we have a list of clusters. We need to find out which cluster corresponds to the object of interest and which ones to discard. Once again we use the position of the pointing finger. However, this position is the pixel coordinates of the finger in the RGB image and the point cloud is an array of points.

The conversion from matrix coordinates to array index is:

- index = y * image_width + x

This conversion only works because the publishing of RGB images and point clouds was registered. This means that the point clouds' points are sorted, in other words, if we apply the formula above to all the points in the point cloud and match their RGB values to the RGB values of the RGB image, they are the same.

With the above formula, we can then obtain the point of the pointing finger's tip in the original point cloud. However, since this point cloud had to be filtered and transformed using the voxel grid algorithm, this point may not exist in the transformed point cloud and consequently in any of the clusters. Because of this, we can't find out which cluster to extract by checking which one contains this point, because this point likely does not exist in any of them. To solve this problem we calculate the euclidean distance of this point to all the points in each cluster. Then the cluster with the smallest mean Euclidean distance is the cluster to extract.

Having the winning cluster with, hopefully, the desired object the remaining step is to extract information that describes this object and save it in a database. Similarly to what was done to the RGB images, descriptors were used. Point Feature Histograms (PFH), Point Feature Histograms RGB (PFHRGB) and SHOTCOLOR were the chosen set to test. These were chosen out of the available descriptors because they proved to have the best results in [ALE12]. The algorithm to extract each descriptor was implemented and applied to the winning cluster, the descriptors were then saved in a database.

The last method tested for information extraction involves obtaining the output of the convolution layers in the pre-trained model. The SURF, SIFT and ORB descriptors extract specific features from the image. These features are hardcoded in the algorithm. However, these algorithms were created in a time were convolutional-neural-networks did not exist. Convolution is the process of applying filters to an image to extract certain information that depends on the filter. The filters are adjusted during the training of the neural-network with the dataset. After convolution, there is usually a down-sampling step to reduce the size of input data. This process is then repeated multiples times resulting in an array of values that are the output of multiple feature extraction steps. This output is what describes the image to the neural-network. These filters were obtained, not from hard code, but by adapting to real-world data. The idea is to obtain a model pre-trained on the 'imageNET' dataset. Then pass an image to the model but, rather than obtaining the output of the last layer, we obtain the output of the layer before the fully connected layers, the output of the feature extraction step. Different network architectures extract different features, we don't know which architecture is superior for this job so they will have to be tested to reach this conclusion.

To find out which method provides the best results, they have to be tested on a dataset. Washington's RGB-D Object Dataset was a perfect fit for this problem [KLF11]. This dataset contains 51 classes of objects. Each image contains only one object and an empty room as the background. Sequences of frames were obtained for each object by moving the camera around, resulting in the capture of images in different angles. For each image captured a depth image in the form of a black and white image was saved, as well as the respective point cloud and a semantic mask. In short, this dataset provides the four types of data required to test all the implemented methods above.

The first experiment consists of analysing the accuracy of the two segmentation models. The images in the dataset are read one by one then, the for each image we read its respective segmentation mask, this is the ground truth mask. We then pass the RGB image to the semantic segmentation model, obtaining what the model believes to be the correct mask. Lastly, we compare the colour of each pixel of the two masks, obtaining the number of identical pixels. We then divide this number by the number of total pixels in the image obtaining a percentage of similarity between the two masks. The average of these percentages is then calculated by summing all the percentages from each image and dividing this sum by the number of images processed. The result is the accuracy of the semantic segmentation model. The dataset has 207662 images, the results are presented in the following table.

Table 11.1: Pixel accuracy comparison between ADE20K and COCO semantic segmentation models on Washington's RGB-D Object Dataset.

Model Name	Pixel Accuracy %
ADE20K	57.69
COCO	81.79

From the results obtained we can conclude that the ADE20K performed quite poorly. Since the masks have binary values, then for each pixel the model can only guess between 2 values. This means that in the worst-case scenario if the model was guessing randomly it would obtain nearly 50% accuracy. ADE20K only performed slightly better than this. On the other hand, COCO performed far better with 81.79% accuracy. This means that if we choose to use a segmentation model in our task we will use COCO and discard ADE20K.

The next experiments consist on identifying the best CNN architecture for feature extraction on RGB images. To perform the experiment, the dataset has to be split into two sets: a training set and a test set. First, the features of the images in the training set are extracted, creating a database. This step simulates the teaching phase of the task. Then, to determine how well the features extracted by the architecture help in identifying the object's correct class, the features from the images in the test set are extracted. Afterwards, each set of features is matched with the sets of features in the database. This step returns the class of object the architecture believes the image belongs to. Lastly, the guessed class is compared with the real class. This process is repeated for all the images in the test set. The result is the number of images the architecture accurately predicted as the test image's class.

Essentially, the results will show which architecture extracts features that have very similar values in images with the same class but are very distinct in images from different classes of objects.

A program was developed to extract the training and test sets from the original dataset. This program receives as its input two numbers. The numbers indicate how many images will be extracted, from each class, to form the train and test sets, respectively. The program then reads the images in a class folder, it indexes them and randomly selects N+M images, where N is the number of images per class for training and M is the number of images per class for testing, placing them in the new dataset folder. It repeats this process for all class folders. With this program, we can quickly change the number of images in the training and test set allowing us to perform experiments with different amounts of data, without having to manually select N+M images ourselves. Additionally, the selection process is random which means that there is no bias towards specific images being included or ignored.

Afterwards, the program responsible for calculating a certain architecture's accuracy was developed. The program starts by reading the training folder and creating a database of features. The training folder contains 51 sub-folders (one per class) and each folder contains the N images obtained in the previous step. Each image will be passed to the CNN and the output of the convolution part, the features, will be saved. The extracted features will be stored in a class list. A class list contains the sets of features extracted from all the images in the class folder. Lastly, the class lists are stored in a global list, which inherently contains all the data. This means that all training data is loaded into memory at once. This is required because the features of each test image will have to be matched with each feature in the training set. This means we are limited to a maximum amount of data in the train set. This amount depends on the available RAM in the computer, for our case its 16 GB, and the number of features extracted by each CNN architecture. In order to not require the train data to be loaded in memory, an alternative could be to extract the features of the images in the train set each time a test image is processed. But this means that the process of loading the database which only happens once if we choose the first approach of loading the entire training set into memory would need to be repeated 51 x M times (once per image in the test set), which would correspond to an exponential increase in execution time.

Next, the images in the test set are processed. The class folders in the test set are sequentially opened and each of its images is processed one at a time. The images are passed to the CNN and its features are extracted, similarly to what was done in the database creation step. Then, the test image's set of features is compared to each set of features in the database. This comparison, which will be explained shortly, returns a prediction for the image's class. If the predicted class is equal to the real class of the test image then a counter of class hits is increased. In the end, this counter is divided by the total number of test images processed which results in the architecture's accuracy.

CNN receive an image as input and output it's class (for classification problems). This means that the convolutional part of the network responsible for extracting features is then connected to a set of fully-connected layers. However, we are interested in obtaining the output of the convolutions not the output of the fully connected layer because we aren't interested in training a network in a static dataset. `Pytorch` library was chosen to fulfil the requirements. First, the library allows us to download a neural-network architecture from a variety of choices. Additionally, we can choose to download a pre-trained model of that network in a data-set, such as ImageNet dataset. It is important to use a pre-trained model in a dataset even though it isn't going to be used for classification because the network was trained for feature extraction. If we simply download architecture with random weights we would obtain poor results since the network has no idea what it is supposed to accomplish. Lastly, it is possible, through some effort, to obtain output from the network's inner layers using `Pytorch`.

The neural-network's architecture determines the size of the set of features extracted every time it receives an image. In other words, the amount of features extracted varies from architecture to architecture but is static no matter the image that is passed to a specific architecture. This fact allows us to use euclidean distance to determine how different or how similar two sets of features are, as both sets have the same amount of numbers. Additionally, one fact that remains consistent between architectures is that the extracted feature set is always a 4-dimensional array. This means that the formula to calculate the distance between two sets of features is the following: $\bar{d}(\bar{s}_1, \bar{s}_2) = \sum_{i=0}^{I} \sum_{j=0}^{J} \sum_{k=0}^{K} \sum_{l=0}^{L} (s_{1ijkl} - s_{2ijkl})^2$. Where d is the distance between the sets of features s_1 and s_2.

Each test image's features are compared with each of the training features extracted in the beginning using the previous formula for each comparison. In order to predict the test image's class, two approaches were tested. One approach is to predict the class to be the same class as the class of the train image with the minimum distance value to the test image. The second approach calculates the average distance of each class to the test image, then the predicted class is the one with the smallest average. These approaches were tested in the same setup, in other words, the same network architecture and same training and test sets. The results are shown in the table below. It is important to mention that for the following two experiments the networks were pre-trained on ImageNet dataset.

Table 11.2: Results of the two tested approaches for a test image class prediction based on the calculated distances in the training set.

Architecture	Training/Test Set Size	Approach Type	# Correct Predictions	%
Resnet50	5/5	Minimum	132/255	51.76
Resnet50	5/5	Min. Class Mean	107/255	41.96
Resnet50	10/50	Minimum	1129/2040	55.34
Resnet50	10/50	Min. Class Mean	948/2040	46.47
VGG16	5/5	Minimum	104/255	40.78
VGG16	5/5	Min. Class Mean	48/255	18.82

From the results presented in table 11.2, we can conclude that regardless of the train set size and neural-network architecture the approach that predicts the class of the train image based on the smallest distance always surpasses the second approach which uses the mean of the classes. Also, the results confirm the suspicion that increasing the train set, even though it increases computational time, increases the accuracy of class predictions.

The next experiment will test different convolutional neural-network architecture's feature extractions. From the previous experiment, the approach that will be used for class prediction will be the one which predicts the class based on the minimum distance. Both the test and training sets will contain 5 images per class. Increasing the test set would provide a more precise accuracy value for each architecture. Additionally, by increasing the train set the accuracy value of each architecture would also increase. However, on one hand, there are time constraints to test a large number of architectures and on the other hand, there are hardware limitations that prevent us from using larger training sets because the RAM isn't big enough to load that much memory. 21 different convolutional neural-network architectures were tested in this setup. These networks were the ones available through `Pytorch`, the library used for obtaining values in middle layers of neural-networks for feature extraction. However, the different CNN architectures have different sizes and also extract a different number of features. In short, some architectures have a higher computational cost than others. This meant that some architectures couldn't be tested on the dataset with all 51 classes due to Random Access Memory (RAM) restrictions. The table 11.3 contains the accuracy results obtained after running the different architectures on the maximum number of classes.

Table 11.3: Results of 21 convolutional neural-network architecture's features on training and test sets with 5 images per class

Architecture	# Correct Predictions	%
DENSENET161	43/50	86.00
DENSENET169	81/95	85.26
DENSENET201	59/75	78.67
DENSENET121	77/100	77.00
Mobilenet	162/255	63.53
Resnet34	148/255	58.04
VGG19	51/90	56.67
RESNET18	143/255	56.08
RESNET101	143/255	56.08
MNASNET1	140/255	54.90
Alexnet	133/255	52.16
Resnet50	132/255	51.76
SQUEEZENET11	116/255	45.49
SHUFFLENET21	108/255	42.35
VGG16	104/255	40.78
SHUFFLENET205	97/255	38.04
Squeezenet	97/255	38.04
Mnasnet05	70/255	27.45
Googlenet	38/255	14.90

From the results presented in table 11.3, we can conclude that the `Densenet` family of architectures out-performed the remaining ones. In particular, `Densenet161` and `Desenet169` achieved levels of accuracy that are good enough for classification. However, these are also the architectures that ran on the least number of classes. Recall from the results obtained during the benchmarking of the person identification algorithm, the accuracy was inversely proportional to the number of classes. This means that other architectures may be superior to densenet when tested on the same number of classes. To test this theory, a second test was performed where all the architectures are tested on the same number of classes. Ten was the maximum number of classes that our computer could handle with the densenet architecture.

Table 11.4: Results of 21 convolutional neural-network architecture's features on 10 classes with 5 points per class in test set

Architecture	# Correct Predictions	%
DenseNet161	46/50	92.00
DenseNet169	45/50	90.00
DenseNet201	42/50	84.00
DenseNet121	41/50	82.00
MobileNet	41/50	82.00
ResNet34	35/50	70.00
AlexNet	34/50	68.00
ResNet152	34/50	68.00
ResNet50	33/50	66.00
ResNet18	33/50	66.00
ResNet101	30/50	60.00
VGG16	30/50	60.00
VGG19	30/50	60.00
MnasNet1	29/50	58.00
SqueezeNet1	26/50	52.00
SqueezeNet11	26/50	52.00
ShuffleNet205	25/50	50.00
ShuffleNet21	24/50	48.00
MnasNet05	23/50	46.00
GoogleNet	17/50	34.00

Even though the average accuracy increased across the board in 11.4, the densenet family of architectures still stands at the top of the table. Additionally, the order of the table is very similar with the main exception being AlexNet architecture which drastically increased in accuracy with the decrease in the number of classes. We may be able to increase the accuracy of classification by combining RGB feature matching with point cloud feature matching, depending on how well those methods perform.

The test was repeated, this time increasing the number of data points in the test set to 100 per class. The results of this test are exposed in table 11.5.

Table 11.5: Results of 21 convolutional neural-network architecture's features on 10 classes with 100 points per class in test set

Architecture	# Correct Predictions	%
DenseNet161	904/1000	90.40
DenseNet169	886/1000	88.60
DenseNet201	844/1000	84.40
DenseNet121	841/1000	84.10
MobileNet	770/1000	77.00
ResNet34	678/1000	67.80
ResNet50	674/1000	67.40
ResNet101	668/1000	66.80
ResNet152	661/1000	66.10
ResNet18	636/1000	63.60
AlexNet	630/1000	63.00
MnasNet1	630/1000	63.00
VGG16	587/1000	58.70
VGG19	569/1000	56.90
SqueezeNet11	566/1000	56.60
ShuffleNet21	547/1000	54.70
SqueezeNet1	502/1000	50.20
ShuffleNet205	487/1000	48.70
MnasNet05	411/1000	41.10
GoogleNet	296/1000	29.60

Even though the order of the architectures in the middle of the table did change in 11.5, the densenet family of architectures is still dominant. Furthermore, the order to the different types of densenets is also the same, though the overall accuracy did slightly decrease.

The next step consists of the identification of which type of point cloud descriptors are going to be tested followed by their implementation. After reading the paper in [Ale12], where 13 different descriptors were tested, PFHRGB and PFH were chosen because they were the best performing. SHOTCOLOR did outperform PFH and its implementation was attempted. Unfortunately, an error kept occurring during its extraction from a point cloud consequently leading to always incorrectly guessing the object's class.

The remaining two types of descriptors were tested using Point Cloud Library (PCL). This library is the go-to library for anything point cloud related because it provides hundreds of classes and functions which ease the development of programs. Unfortunately, it is implemented in c++ language. This fact not only requires learning this language but also requires the creation of a way to call these c++ programs from the python scripts. A solution was found by making use of ROS' frameworks. The c++ program publishes messages to a topic, these messages are then received by the python program which is subscribed to this topic.

Two programs, similar to what was done for the RGB descriptors, were developed to test the accuracy of the point cloud descriptors on the Washingon's dataset. The first splits the dataset into training and test sets, only this time the program only works with the point cloud files rather than the RGB images.

The second program, similar to what was done for the RGB descriptors, is responsible for calculating a point cloud descriptor accuracy on the test set based on the information in the training set, only this time it had to be written in c++. The logic of this program is similar to the one developed for the RGB descriptors. The program reads the training set, extracts the descriptors from each point cloud, places it in a class list which is then placed in a global list. The program then reads each point cloud in the test set, extracts its descriptors, compares them with the ones in the global list obtaining a predicted class and lastly compares the predicted class with the real class to calculate the accuracy of the method. In short, the differences between this program and the one developed for the RGB descriptors, apart from the fact that it is written in c++, is the way that the descriptors are extracted as well as the way they are compared with each other. The code responsible for extracting each type of descriptor was adapted from the official PCL's documentation.

The cloud is first loaded into memory. Then to reduce the number of points to be processed, the cloud is filtered using a Voxel Grid. Next, the normals of the points in the filtered cloud are calculated. The filtered cloud and normals are passed to the function present in the PCL which computes the features, one of the two types of descriptors.

The number of points in the calculated descriptor depends on the number of points of the point cloud. In a nutshell, unlike the RGB descriptor method where the number of points extracted only depended on the CNN's architecture and not the images, the point cloud's descriptors vary with the size of the input cloud. This affects the way the comparison of the two descriptors is made. Since two descriptors may and likely have different numbers of points, the euclidean's distance formula can't be applied as there isn't a direct one to one match between the indexes of each descriptor. For this reason, a different formula that calculates the distance between two sets with an unequal amount of points was required. The chosen formula was Hausdorff's distance [ALE13].
$D(A, B) = max\{sup\{d(a, B) \mid a \in A\}, sup\{d(b, A) \mid b \in B\}\}$.
Where $d(a, B) = min\{ed(a_i, b_i), i = 1, ..., \mid B \mid\}$ and ed is the euclidean distance between two points.

The program ran on a dataset with 10 point clouds for training and 40 for test, per class. The results are shown in the table 11.6.

Table 11.6: Accuracy of point cloud descriptors on the Washington dataset using the Hausdorff distance

Descriptor Type	# Correct Predictions	%
FPFH	346/2040	16.96
PFHRGB	374/2040	18.33

The table shows that both types of descriptors presented poor accuracy values. This means that this method can't be used as a standalone. In other words, one can't reliably obtain the correct class of the object in the robot's camera by obtaining the point cloud and passing it through either of these methods. However, these methods may still be useful when combined with the RGB descriptors by potentially increasing their accuracy.

The next step will then test the accuracy of the combination of different methods to see if the accuracy obtained from the densenet architecture can be improved further. This means the combination of methods that potentially are written in different programming languages, RGB descriptors are written in python whilst PC descriptors are written in c++. One additional setback is the fact that if we want to test the accuracy of a certain method X combined with different other methods the execution of the method X will have to be repeated each time. In other to solve all of these problems, the programs are changed to, instead of calculating the accuracy of the method on the test set, saving the output to a text file as a table. This table will have one row per file in the test set while each row will have as many columns as the number of classes, Washington's dataset has 51, with an additional column that contains the name of the text file. A cell in row R and column C will contain the minimum distance of the text file's descriptor of row R to the descriptors of each of the train files of class C.

Before proceeding further, it is important to identify a problem. If we want to test the accuracy of the combination of an RGB method with a PC method, then we need to make sure that the point cloud files in the point cloud dataset match with the RGB images in the RGB dataset. Otherwise, the distances would not match. This meant the creation of a program which creates both RGB and point cloud datasets where their files match with each other.

A program was then created to calculate the accuracy of the combination of two methods, through the combination of their respective tables. The program receives the names of the two tables and proceeds to read the respective files loading each table into a two-dimensional array. Then, each of the tables is normalized. This is done because the values of distance in different tables likely are on different scales. If one table has values in the thousands while the other has values in the hundreds then the first table will have a higher impact on the conclusions. Therefore, the tables have to be converted into the same scale through normalization. Normalization was done by first calculating the minimum and maximum values in the table. We will call amplitude the difference between the maximum value and the minimum value. Then, each value in the table is subtracted by the minimum value, this makes the range of values start at 0 since no number can be negative as there is no number smaller than the minimum. Lastly, the numbers are divided by the amplitude. Since the range was varying from zero to amplitude, by dividing the numbers by the amplitude, the range now varies from zero to one. This way, all the values in the tables keep their proportions to each other but both tables now have the same range of values.

Afterwards, the program creates a table which is the combination of the two normalized input tables. The program iterates over the first table row by row. It reads the name of the text file associated to the row then finds the index of the row associated with the same test file in the second table. This is necessary as the files may not be read in the same order in different executions meaning that the indexes of rows of different tables don't match. Then each index of the row's columns is iterated over.

Each table has an associated weight, this value indicates how much influence the respective table will have on the creation of the combined table. The weight of a table will be the accuracy of the respective method obtained during the previous tests. A cell's value in the combined table will be the result of the sum of the multiplication of the cell values of each table by their respective weight divided by the sum of the weights. The formula is the following:

$$CT_{rc} = (T_{1rc} * W_1 + T_{2rc} * W_2)/(W_1 + W_2)$$

Where CT is the table that resulted from the combination of tables T_1 and T_2, r and c are, respectively, the row and column indexes and lastly W_1 and W_2 are the weights associated to the tables.

With the previous algorithm, we can obtain a table which is the result of the combination of two tables that represent two methods. The last step left is to calculate the accuracy represented by this table to know if the combination of the two methods constitutes an improvement. To achieve this, each row of the table is iterated one by one. The real class of the file associated with the row is identified. Since we assured that the class folders were read alphabetically and the number of test files per class is static, we can simply divide the row's index by the number of test files per class to obtain the real class index. The predicted class is the column index of the cell with the minimum value in the row, as this is the class with the minimum distance to the test file. Similarly to what was done in previous algorithms, a counter variable keeps track of the number of times the predicted class equalled the real class. In the end, the accuracy resulting from the combination of the two methods is obtained by dividing this counter variable by the number of rows in the table (number of test files).

All possible combinations were tested. Since there are over 200 possible combinations, only the combinations of N architectures which increased the maximum accuracy of the N-1 architectures are listed below.

- Alexnet + DENSENET169 - 94 %
- DENSENET121 + DENSENET161.TXT + PFHRGB + VGG16 - 96 %
- DENSENET169 + DENSENET201 + PFHRGB + VGG16 - 96%

The results refer to the evaluation of accuracy on the dataset with 5 data points per class in the test set. This is important to refer because the combination of the tables that were obtained from the execution on the dataset with 100 data points per class didn't show any combination that improved accuracy. Therefore, since this dataset has a higher sample size, the conclusion will be taken from its results. Consequently, no combination of architectures will be used. Instead, the highest performing architecture, DENSENET161, will be used for feature extraction in the task.

11.1.4 Algorithm

After studying and testing different approaches and reaching definitive conclusions it is time to integrate all the separate tools into one program that allows the robot to perform the learning task. The program starts by checking if the database directory exists and, in case it doesn't, creates it. Next, the program starts the AMCL process (compatible with follower) and creates a subscriber to receive images from the camera. This subscriber's callback function simply receives and stores the latest image. Then a loop is started which contains the main logic of the task. First, the program starts the follower process which causes the robot to follow the human. The program uses the speech recognition module to constantly recognize speech. Once this module recognizes that the human said ”stop”, the program stops the follower process and the robot stops following the human. Then, the program waits for the human to say the object's name using the ”this is” logic explained previously. At this point, the robot knows the object's name and should be able to see it. The last step is to extract and save the object's features. The program tells the human to point to the object and constantly checks for the existence of a pointing finger in the frame. Once it detects the finger, the program saves its pixel position. Next, the program tells the human to stop pointing and waits for the finger to be removed from the frame. Afterwards, the image is segmented using the COCO model and the pointing finger's position. Lastly, the features of the image are extracted and stored in the database in a folder whose name corresponds to the object's name. At this point, the robot asks the human if the task is finished. If the answer is affirmative then the program stops, else the cycle is repeated.

11.2 Identifying changes on the map

In the second part of the task, the part that is evaluated by the competition, the robot will have to use the information gathered from the previous step to identify changes that were since made to the environment. The implementation of this step was much quicker because all of the software developed. The robot receives as input a semantic map, generated in the previous step. If it receives a different type of file the program can identify this and throws an error. At the beginning of its execution, the program starts AMCL. This achieves two things: the program can create a subscriber to receive images from the camera as well as be able to send navigation goals to move the robot.

The semantic map is read 10 lines at a time since every 10 lines represent an object. For each object in the semantic map, the program will use the navigation module and move the robot to the object's location. It repeats the navigation goal until it is successful. Because the map contains both the position and orientation of the robot when the object's information was captured, when the robot achieves these values it should be facing the object (if it is still there). The only thing left to do is for the robot to verify if the object is still present. To achieve this, the last image captured by the camera is used and its descriptors are extracted. Then the descriptors stored in the database correspondent to the object in question are read into memory.

Next, the distance between the new descriptor and each of the descriptors in the database is calculated. If the average distance is smaller than a certain threshold then the object is still present, else the object is missing. In either case, the robot speaks the conclusion it reached through its speech module. Then the cycle is repeated until there are no more objects in the semantic map. The threshold's chosen value was 1.1. This value was reached through experimentation. The robot would move to locations where the object was still present and the distance calculated stored. Similarly, to find the distance when the object is not present, the robot was moved to locations where the object was missing. This value was the average between the maximum value obtained in the first scenario and the minimum value obtained in the second scenario.

11.3 Conclusion

This chapter explained the implementation of the 'Getting to Know my Home' task. It consists of two parts. The first part involves the creation of a semantic map. A simpler approach could have been taken as the competition doesn't specify how the map should be built. However, the goal was to create an approach which allowed a quick and simple method of performing this step as its only a setup for the actual task. Furthermore, creating an approach which continually and dynamically increases the robots knowledge of objects can be extended to other tasks, for example, the 'Speech Object and Person Recognition' task, but also potential future ones. To achieve this, many descriptor extraction methods were tested. Results show that from the architectures tested, DENSENET161 provides the highest accuracy. Afterwards, an algorithm was implemented where the robot follows the human around the environment and is taught the object's names and locations one at a time. Then an algorithm which reads a semantic map generated by the previous step was implemented. The robot goes to each location specified on the map, extracts the descriptors of the images captured by the camera, compares them to the database and concludes if the object is still present or is missing.

Chapter 12

Conclusions

This chapter contains an overview of what was accomplished from the development of this thesis. It will answer the question if all that was initially stated to be achieved ended up being accomplished. Additionally, ideas on what parts can be improved further, as well as how they can be improved will be stated.

12.1 Foundations

The first thing that had to be done was to update the robot's software. Both ROS and Ubuntu are regularly updated. This means that after the conclusion of this project, both will have to be updated once again. The turtlebot specific packages were installed from source. This means that they won't have to be reinstalled when ROS is updated. Afterwards, ideas on how to ease the integration and development of new software were thought of. The goal was to make every bit of functionality a module so that it can simply be imported and used by the programs that require it. This prevented the functionality from having to be re-written, which means that bug fixing and/or updating, only needs to be performed in the module itself, rather than in every program that implemented such functionality. These, however, are just guidelines, future software should follow them to guarantee integration and scalability, rather than reverting to the initial state before the development of the thesis, with scattered projects. Additionally, a problem was identified that is a consequence of running programs that make use of ROS' functionalities, simultaneously. The implementation of the process controller class prevented crashes that occurred due to the concurrent execution of ROS processes which use identical nodes. Additionally, it takes care of starting dependencies automatically rather than having to re-write the starting logic in every program that requires a certain process. Furthermore, the initialization and stopping of each process are controlled by the class, allowing synchronization with the program's instructions. In a nutshell, the implementation of this class allows other programs to start the required processes in a much less difficult and verbose manner and without having to worry about potentially crashing. However, each process has to be specified in the class through the creation of a start and stop function. There are many available ROS processes, and the number keeps increasing as time goes by. This means that new programs may require the execution of a process that isn't available in the process controller class. Therefore, the programmer will have to update this class through the creation of the respective start and stop functions. The creation of both these functions is simple and fast due to the examples of other processes and due to the explanation of how to write such functions in the respective chapter.

12.2 Robotic Tasks

The second goal of the project was to implement as many tasks as possible with the available time to develop the thesis. For this purpose, seven robot tasks were identified. All seven tasks ended up being implemented. Furthermore, the creation of these tasks required the creation of functionalities which were turned into modules which means they can be used by future tasks, consequently cutting the amount of work required. Furthermore, this meant that this single project implemented more tasks than any other so far, corresponding to a gigantic step to achieve the goals of participating in robotic competitions as well as having an autonomous robot roaming around campus.

12.2.1 Help me Carry

This task had been implemented by a colleague. Improvements were performed directly on his implementation. The code was restructured to fit the modularity goal. The RQ code method of communicating with the robot was replaced with speech recognition. Lastly, the depth detection method of which the robot uses to get closer to the human was improved, consequently making it more reliant and resistant to measurement errors from the camera.

12.2.2 Restaurant

To implement this task, a way of detecting when a person was waving at the robot had to be created. After a couple of attempts, a working solution was created. However, the solution has its problems. Due to the inaccuracies of hand and face detection modules, the approach often fails to detect a handwave and requires the human to handwave for a longer than what would be ideal. Additionally, the approach does sometimes inaccurately detect a handwave when there was no such thing. Creating or finding better hand and face detectors would improve the current algorithm. On the other hand, the approach which discussed the creation of a dataset that classifies hand gestures could be implemented in a dedicated project. It was not attempted as the allocated time for the implementation of the task was running out and there wasn't a guarantee that the approach would work.

12.2.3 Elevator

This task could have been statically implemented by generating a map of the elevator before-hand, then specifically finding the coordinates that the robot must move to. However, the goal of the implementation was to develop a more dynamic solution to the problem. The final implementation allows the robot to enter and exit potentially any type of elevator.

12.2.4 Speech Object and Person Recognition

This task required the robot to answer a question and to correctly name an object being shown to it. To solve the first challenge two approaches were developed. One uses Wikipedia's API to answer the question. This approach works rather well but it isn't perfect because there is no guarantee that Wikipedia has the answer for the question, nor that the answer to the question is in the first few sentences of the page that had the biggest match with the question. Furthermore, the approach only works if the robot has internet access to communicate with the API. Therefore, an offline approach was implemented which gathered thousands of questions and answers in a text file. This approach is rudimentary as the likelihood of a question being in the text file is small. The conclusion is that the online method is a fine implementation but there is a large margin for improvement through an allocation of a project with this goal. To solve the second problem, a dataset was created. This dataset was obtained from the list of items present in the competition's rule-book. However, this list may be updated in the future which will require the dataset to also be updated accordingly. Furthermore, to provide the robot with the ability of object recognition outside of the scope of the competition, a new dataset was explored. However, after developing the 'Getting to know my Home' task, this step could be replaced with the object learning process of this task. This is because it allows for infinite expansion, while the model was obtained from a dataset with a very small number of objects.

12.2.5 Welcoming Visitors

A working implementation of this task was achieved through the integration of a face recognition module present in a python package. However, the face recognition state-of-the-art keeps improving meaning that this model can be replaced by a better one in the future. Such change is simple and quick to perform since all that is necessary is to change the model in the function responsible for extracting the face descriptors.

12.2.6 Catering for Granny Annie

This task required the robot to understand three specific commands. The recognition of these commands was implemented with the addition of allowing the human to change certain variable's values in the speech. Although this task was implemented successfully, its implementation does not allow the rapid integration of more commands. This is because the command recognition was implemented through the creation of an automata for each type of command. Even though this solution works, it still requires the design and implementation of an automata for each new command.

12.2.7 Getting to Know my Home

This task required the creation of a semantic map of the environment. The robot later compares the information on the map with the state of the environment to spot changes. The rules of the competition don't restrict the way the map should be created. The chosen approach aimed to be as robust, dynamic, and lasting as possible. The robot continually expands its knowledge of objects by following the human around the room and being taught the objects' names. The learning is based on the features extracted from convolution. Different CNNs extract different features. This required the evaluation of the quality of features extracted from different architectures to identify which one should be used for this particular task. Testing was also done to verify if the accuracy could be increased through the combination of multiple types of features. Results showed that the densenet family of architectures had the highest accuracy. However, since there are infinite possibilities for CNN architectures, and there are already more architectures than the ones that were tested, there is a high chance that the accuracy of the method could be improved by using a different architecture. Nonetheless, this change implies the simple replacement of architecture in the feature extraction part of the code. Therefore, the results were a success because a way of dynamically adding objects to the robot's knowledge was found.

12.3 General Conclusions

The development of this thesis was a difficult yet enjoyable challenge. All the initial goals were achieved. This means that the robot is now able to perform seven individual robotic tasks. Furthermore, the implementation of these tasks required the implementation of blocks of code, modules, which can be re-used by future tasks. This fact, combined with the software created for easing the development and integration of new software, represents a solid foundation for the creation of new robotic tasks. Tackling each task and later achieving the solutions required using tools from multiple branches of the area of artificial intelligence. Therefore, this thesis allowed me to learn about multiple sub-areas of artificial intelligence, broadening my knowledge rather than specializing in a specific problem.

Bibliography

[AAA+18] Aamir Ahmad, Francesco Amigoni, Iman Awaad, Jakob Berghofer, Rainer Bischo, Andrea Bonarini, Rhama Dwiputra, Giulio Fontana, Frederik Hegger, Nico Hochgeschwender, Luca Iocchi, Pedro Kraetzschmar, Gerhard Lima, Matteo Matteucci, Daniele Nardi, Pedro Miraldo, Viola Schiaonati, Sven Schneider, Meysam Basiri, Emanuele Bastianelli, Pedro Resende, and João Mendes. Erl consumer: European robotic league for consumer service robots. `http://www.https://www.eu-robotics.net/robotics_league/upload/documents-2018/ERL_Consumer_10092018.pdf`, 2018. 11

[ALE12] Luís A. Alexandre. 3D descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October 2012. 76, 82

[ALE13] Luís A. Alexandre. Set distance functions for 3D object recognition. In *18th Iberoamerican Congress on Pattern Recognition*, volume LNCS 8258 of *Lecture Notes in Computer Science*, pages 57–64, Havana, Cuba, November 2013. Springer. 83

[ALST16] A.A. Abdulla, Hui Liu, Norbert Stoll, and Kerstin Thurow. A new robust method for mobile robot multifloor navigationin in distributed life science laboratories. *Journal of Control Science and Engineering*, 2016, 08 2016. 51, 53

[ATC+12] R. Amano, K. Takahashi, T. Cho, K. Kobayashi, K. Watanabe, and Y. Kurihara. Development of automatic elevator navigation algorithm for jaus compliant mobile robot. In *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, pages 1633–1638, Nov 2012. 51, 52

[FER18] António Fernandes. Preparing a robot for the robocup@home competition. 06 2018. 29

[KAO07] Jeong-Gwan Kang, Su-Yong An, and Se-Young Oh. Navigation strategy for the service robot in the elevator environment. pages 1092 – 1097, 11 2007. 51, 52

[KLF11] Xiaofeng Ren Kevin Lai, Liefeng Bo and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset, May 2011. 77

[LM17] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition, 2017. 48

[LMB⁺14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, ABS/1405.0312, 2014. Available from: `http://arxiv.org/abs/1405.0312`. 74

[MRW⁺19] Mauricio Matamoros, Caleb Rascon, Sven Wachsmuth, Alexander William Moriarty, Johannes Kummert, Justin Hart, Sammy Pfeiffer, Matthijs van der Brugh, and Maxime St-Pierre. Robocup@home 2019: Rules and regulations (draft). `http://www.robocupathome.org/rules/2019_rulebook.pdf`, 2019. 9

[PAI19] Luis Pais. Integrating new functionality into a TURTLEBOT2. 06 2019. 40

[TSG⁺13] D. Troniak, J. Sattar, A. Gupta, J. J. Little, W. Chan, E. Calisgan, E. Croft, and M. Van der Loos. Charlie rides the elevator – integrating vision, navigation and manipulation towards multi-floor robot locomotion. In *2013 International Conference on Computer and Robot Vision*, pages 1–8, May 2013. 51, 52

[Uni94] University of Essex. Faces94 [online]. 1994. Available from: `https://cswww.essex.ac.uk/mv/allfaces/faces94.html`. 67

[ZCL19] J. Zhao, Y. Chen, and Y. Lou. A human-aware robotic system for mobile robot navigating in multi-floor building with elevator. In *2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pages 178–183, Aug 2019. 52

[ZZP⁺17] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 74