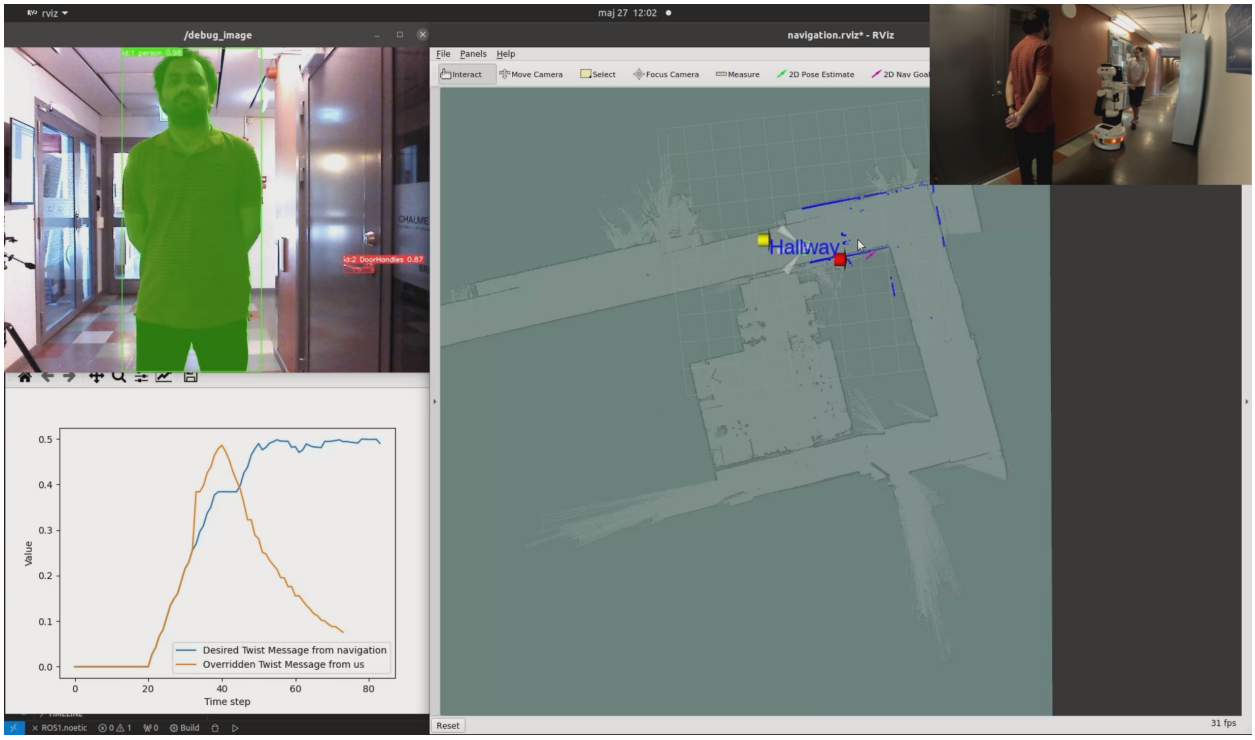
CHALMERS
UNIVERSITY OF TECHNOLOGY



Perception & Decision-Making with YOLO & ChatGPT Integration for Intelligent Robot Navigation

Master's thesis in Systems, Control & Mechatronics

Oscar Hjertqvist, Suraj Krishnamurthi

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Perception and Decision-making with YOLO and ChatGPT Integration for Intelligent Robot Navigation

Oscar Hjertqvist
Suraj Krishnamurthi

Department of Electrical engineering
Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2024

Perception and Decision-making with YOLO and ChatGPT Integration for Intelligent Robot Navigation
Oscar Hjertqvist Suraj Krishnamurthi

© Oscar Hjertqvist, 2024. © Suraj Krishnamurthi, 2024.

Supervisor: Karinne Ramirez-Amaro, Electrical Engineering
Examiner: Karinne Ramirez-Amaro, Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The TIAGo robot present in the lab which was worked on throughout the thesis

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Perception and Decision-making with YOLO and ChatGPT Integration for Intelligent Robot Navigation
Oscar Hjertqvist, Suraj Krishnamurthi
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In the eld of autonomous robots, perception and navigation in dynamic environments are important for ensuring safety. The motivation for this work is to improve the safety of a robot navigating in di erent dynamic environments, such as in a lab with a human moving around. This thesis presents a system which integrates object detection into a robot which in turn allows environmental perception in the form of recognizing objects or humans in its surroundings. This allows the robot to make decisions on how fast it should navigate depending on the room it is located in and the movement of humans. The main questions to be addressed include how well can the robot perceive its surrounding environment and decide how to navigate as well as how it should react to humans based on real-time data.

The implementation involves using the YOLO (You Only Look Once) algorithm for object detection and the ChatGPT API for determining the room's location based on the detected objects. By utilizing the occupancy grid map, which is a map that is divided into many small cells where each cell can either be occupied, non-occupied or non-de ned, along with laser scan data and an RGB-D camera, the robot can identify walls and objects in its path. Additionally, it can calculate the speed and position of a human relative to the grid map. Furthermore, by using ChatGPT it can generate speech responses dependent on the room and situation that the robot is in and voice these responses to show its intent.

The results of this thesis include navigation testing in di erent rooms and performance metrics of the YOLO object recognition model. These results clearly showed that the robot could detect objects and humans to a good level of precision in its surroundings with a trained YOLO model with a precision of 87%. The ChatGPT API accurately identi ed the room 95.6% of the time when queried with the same set of objects. The navigation performance was adequate as the robot could recognize humans and adjust its movement speed based on the situation and location. Though the laser range nder occasionally returned slightly erroneous data, it detected things that weren't actually there, which made the navigation not perform perfectly.

Therefore, it can be concluded that the integration of perception and decision-making have improved the safety of the robot while navigating and thereby increasing the capability of navigating through a complex and dynamic environments.

Keywords: TIAGo, Robot, Navigation, Perception, Decision-making, YOLO, Object Detection, Object Segmentation, ChatGPT API, AI.

Acknowledgements

As a team of 2 students, it gives us great pleasure and immense satisfaction in presenting this Master's thesis report which became possible because of a great guidance, support and timely suggestions from our supervisor and examiner Karinne Ramirez-Amaro. It was very valuable for us to gain some knowledge in the domain of robotics from her.

Being able to showcase the di erent things implemented by us to other people who came to visit the lab under the suggestion of our examiner made us very happy to know that our work is getting recognized by others and having the opportunity to meet di erent people from diverse background made us know how involved people want to be in the domain of robotics which has gained popularity in recent times.

We are also very thankful to our friends and colleagues for their continuous support to us in challenging times.

Last, but not the least, we would sincerely like to thank our family members for their unconditional love and support which directly or indirectly contributed towards the creation of this thesis.

Oscar Hjertvist, Suraj Krishnamurthi, Gothenburg, May 2024

Glossary

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Arti cial Intelligence
ChatGPT API	ChatGPT Application Programming Interface
FOV	Field of View
ROS	Robot Operating System
AMCL	Adaptive Monte Carlo Localization
SLAM	Simultaneous Localization and Mapping
YOLO	You Only Look Once
LLM	Large Language Model
TTS	Text To Speech

Contents

Glossary ix

List of Figures xiii

1 Introduction 1
1.1 Background 1
1.2 Purpose 5
1.3 Research Questions 5
1.4 Demarcations 6
1.5 Related work 7

2 Theory 9
2.1 Object detection using YOLO 9
2.2 Robot Communication - ROS 10
2.3 Mapping & Localization 12
2.4 Quaternions & Euler Angles 13

3 Methods 15
3.1 Mapping with ROS 15
3.2 Training the YOLO Model 18
3.3 Calculating Speed of the Robot and Human 20
3.3.1 Calculation of Robot Speed 20
3.3.2 Calculation of angle between Robot and Human 20
3.3.3 Calculation of Human Speed 22
3.4 Ontology-based reasoning 23
3.5 Implementing ROS Nodes 24
3.5.1 Creating instances of objects and room detection Node 28
3.5.2 Developing the Wall Detection Node 32
3.5.3 Testing the Decision-Making Node 33
3.5.4 ChatGPT API Implementation 35

4 Results 37
4.1 Identifying Possible Sensors for ROS Integration 37
4.2 Analyzing Mapping Results 38
4.3 Evaluating the YOLO Model 40

4.3.1 Limitations observed after Data Training 43
4.4 Assessing Instance Creation 43
4.5 Evaluating the Room Detection 45
4.5.1 Analyzing ChatGPT API Results 45
4.5.1.1 Heatmap Plot Analysis 49
4.6 Evaluating the Wall Detection Node 54
4.7 Testing the Decision-Making Node 55

5 Conclusion and further work 61
5.1 Conclusion 61
5.2 Further work 62

A Appendix 1 I
A.1 Default Parameters for YOLO I
A.2 Decision for Lab II
A.3 Decision for living room III
A.4 Decision for Hallway IV

List of Figures

1.1 The TIAGo robot and its sensors, with exception for sonar, that was used for the thesis. 2
1.2 The SONAR sensor present on the backside of TIAGo robot 3
1.3 At the top (a) the color feed of the camera can be viewed. At the bottom (b), the depth feed given by the RGB-D camera can be seen. 4
1.4 The decision tree from the paper "Perception and Decision Making for Intelligent Robots [4]" that describes each of the actions that the TIAGo robot takes depending on various factors. 7
1.5 ProgPrompt's work ow for generating and executing plan for robot tasks with large language models (LLMs) like GPT-3 from the paper "ProgPrompt: Generating Situated Robot Task Plans using Large Language Models [8]" 8

2.1 A simple representation of how nodes can communicate with each other using topics. 11
2.2 A simple representation of how nodes can communicate with each other using services. The two gures 2.2a and 2.2b show the request and response respectively. 11
2.3 The robot navigating in a corridor where the cluster of red arrows right in front of the robot represent the di erent particles that have converged. From the paper "Perception and decision making for intelligent robots [4] " 12
2.4 A TIAGo robot in a warehouse with a multitude of particles that represent possible poses of the robot. Image was originally from the TIAGo tutorial "Localization and path planning" by Jordi Pages [24]. 13
2.5 Euler angle representation on top of the TIAGo robot. 14

3.1 Pipeline showing di erent sections present in the chapter of methods 15
3.2 Door handle annotated using robo ow 18
3.3 Number lock annotated using robo ow 19
3.4 Calculation of Angle to a green sphere from the RGB-D camera of the robot. 21
3.5 The ontology created with the di erent objects created as classes. . . 24

3.6 Block Diagram of ROS nodes used in the thesis. Blue circles represent nodes, and green arrows show topic communications. For example, the Wall Detection node subscribes to the topics /map and /laser, then publishes to the topic /wall_distance, which is subscribed to by the Decision-Making node. Text with no forward slash represents inputs that aren't topics, such as the response from ChatGPT. 25

4.1 The visualization of the sonar on the robot when detecting a human. 38
4.2 Map of the lab as well as the surrounding hallway 38
4.3 Map of the living room and connecting hallway. 39
4.4 Map of the lab as well as the surrounding hallway with actual objects circled in red and objects that do not exist in yellow 39
4.5 The two gures show (a) the YOLO model recognizing objects and (b) the same view but with a human as well. 40
4.6 Normalized Confusion Matrix obtained after the rst training 41
4.7 Normalized Confusion Matrix obtained after the second training . . . 41
4.8 Example of data predicted wrongly 43
4.9 The live camera feed as well as the map which includes di erent objects that have been seen and the room that the robot thinks its in. 44
4.10 Normalized bar graph distribution for bench across di erent rooms . 46
4.11 Normalized bar graph distribution for chair across di erent rooms . . 46
4.12 Normalized bar graph distribution for door handle across di erent rooms 47
4.13 Normalized bar graph distribution for laptop across di erent rooms . 48
4.14 Normalized bar graph distribution for number locks across di erent rooms 48
4.15 Normalized bar graph distribution for tables across di erent rooms . 49
4.16 Heatmap of di erent object combinations and subsequent room that ChatGPT says it is given a less speci c query. 53
4.17 Heatmap of di erent object combinations and subsequent room that ChatGPT says it is given another more speci c query. 53
4.18 Visualization of the laser range nder when robot is facing a wall with no obstacle in robot's path 54
4.19 Visualization of the laser range nder when robot is facing the wall with foot of a human as an obstacle in robot's path 55
4.20 The gure shows the computer screen when a test is occurring. Showing the camera feed, graph of the speed of the robot and the map. 56
4.21 The desired and actual speed when the robot is navigating in the lab and a human goes into and then out of view 2 times. 56
4.22 The desired and actual speed when the robot is navigating in the hallway and a human is standing right in front of the robot. 57
4.23 The desired and actual speed when the robot is navigating in the hallway and a human is rst running next to the robot and then stands in front of it, but to the side. 58

4.24 The desired and actual speed when the robot is navigating in the living room. 59

A.1 List of default parameters used in the YOLO training I

1

Introduction

There have been some tremendous advancements made in the eld of robotics and it has become possible for the robots to share spaces with humans ranging from factories to o ces to even in household environments. Since this is an age where new technology is getting developed almost every week, the robots sharing spaces with humans is expected to gradually increase in the following years [1]. Due to the fact the robots are now sharing spaces with humans, it has become essential and important to develop a decision-making system for the robot to minimize the risk of collision between them or anywhere else with any particular object. In order for successful deployment, there is a need for the robot to excel in dynamic environments so that it can avoid obstacles such as humans, chairs, tables or any furniture items present in its path and reach its destination without collision. The primary decisions that the robot will make include modifying the velocity at which it navigates at, as well as stopping when necessary, depending on the observed environment.

The main goal of this thesis is to develop a real-time decision-making system for robot navigation in dynamic environments. This is to be done by detecting objects using YOLO and integrating ChatGPT for room identi cation which depends on those detected objects. This will allow the robot to make room-dependent decisions regarding its velocity while navigating, limiting the risk of collision with potential objects.

1.1 Background

The robot used in this thesis is the TIAGo 101c robot manipulator, which can be seen in gure 1.1. This robot manipulator is a test bench to try out the di erent functionalities that are available on the robot. The TIAGo Robot has several sensors [2] which can be seen in the following list and their location on the robot can be viewed in gures 1.1 and 1.2 :

- A laser range- nder in the front of the base.
- Sonars in the back of the base.
- IMU in the base.

- Motors current feedback in arm/wheels.
- Stereo microphones and speakers in the torso.
- Force & torque sensors in the wrist.
- RGB-D camera on the head.

For this thesis in particular, the laser range nder, odometry data from the wheels, the RDB-D camera and the speakers are the features that are most used in this thesis.

Figure 1.1: The TIAGo robot and its sensors, with exception for sonar, that was used for the thesis.

Figure 1.2: The SONAR sensor present on the backside of TIAGo robot

The camera that is integrated into the head of the TIAGo robot is an Astra S from Orbecc [3]. The camera has a FOV (Field Of View) of 60° horizontally and a live video resolution feed in both depth and color of 640x480 pixels at 30 fps and the depth camera is recommended to be used with a distance between 0.4m and 2m [3]. In figure 1.3, the two different camera feeds can be viewed as pictures taken when the camera was being used in real-time.

(a) Live color feed from the RGB-D camera.

(b) Live depth feed from the RGB-D camera.

Figure 1.3: At the top (a) the color feed of the camera can be viewed. At the bottom (b), the depth feed given by the RGB-D camera can be seen.

The laser rangefinder on the TIAGo robot is a SICK TIM561-2050101 laser rangefinder which has a range of 0.05 - 10m with a FOV of 180°. It continuously scans in front of the robot with a frequency of 15 Hz [3]. The sonar on the TIAGo robot is a set of three separate ultrasonic sensors by Devantech SRF05. These use a frequency of 40 kHz and have a measuring distance of 0.03 - 1m [3]. Both of these sensors are used by the robot and the output of them can be accessed by subscribing to their topics using ROS. How that specifically works will be explored further in the theory

section.

Furthermore, a basic navigation package is already implemented on the robot [2]. Which contains a localization method for the TIAGo robot to localize itself based on a map. Though this localization method only places the robot on the map, it does not detect which room it may be in. This navigation package does not recognize any objects, nor can it classify them, it can only detect if there is anything there using its RGB-D camera and laser range nder. Additionally, it does not slow down or stop if there are humans moving around.

1.2 Purpose

The main purpose of this thesis is to develop and showcase the real-time decisions of a robot by combining object detection and navigation to improve safe movement in a dynamic environments. This involves using the YOLO (You Only Look Once) algorithm to train a custom dataset, enabling the robot to accurately recognize various objects in its surroundings such as chairs, tables, benches, laptops, door handles, number locks as well as humans. The object recognition is important for the robot to navigate safely, avoiding obstacles and minimizing collision risks.

Furthermore, this thesis intends to integrate the ChatGPT API to interpret a list of nearby detected objects, helping the robot identify which room it is currently located in, such as if it is in a lab, living room, or hallway. This localization is important for room-dependent decisions. For example, in a lab, the robot should make the decision to move more slowly due to di erent objects lying on the oor and humans walking around.

Moreover, to improve safety, the robot should interact with the humans in front of it by voicing out its intentions and the reasons for those intentions. Subsequently, the human will know what action the robot is about to take. This will help in minimizing the risk of collision thereby, improving the safety of both human and the robot.

1.3 Research Questions

Two main areas were investigated in this thesis which were decision-making and sensor fusion.

Decision-making : How can the decision-making system be enhanced to enable the TIAGo robot to respond to complex scenarios?

1. What decision-making algorithms are suitable for handling dynamic scenarios involving humans and other objects in the robot's path?

2. How can the decision-making be implemented to allow the robot to overtake a human? That is, to move past a human if there is no risk of collision and

stop if there is a risk.

3. How can the robot infer its position among di erent rooms depending on the objects it perceives, for example, if it perceives a bench, it should infer that it is in the living room?

4. How can the decision-making be implemented for the robot if it either perceives a human in its path that then moves out of the way or a human that is not in the path but moves in its way?

Sensor fusion : How can sensor fusion in the TIAGo robot's intelligent navigation system be adjusted so that it can decide which sensor is better to use depending on the situation such as using di erent sensors, among the laser range nder, RGB-D camera and the sonar.

1. What criteria, based on which room the TIAGo robot is located in, will be used for the need of switching or merging the sensors present in the robot?

2. Which of the three di erent sensors are to be used in this project? Lidar, sonar, RGB-D camera or a combination of them?

3. Does the TIAGo robot navigate better when using the laser range nder as well as the sonar in comparison to only using the RGB-D camera?

4. How can the fusion of sensors be used for detecting a change in the trajectory of a human?

1.4 Demarcations

As this thesis will primarily focus on the decision-making and using the di erent sensors some parts will not be investigated.

- The robotic arm as well as the gripper on the TIAGo robot will not be investigated or used.
- The vertically extendable torso will not be used, the height of the robot will be kept constant
- The scenario where there are multiple humans at once will not be investigated.
- The microphone on the robot will not be used, however the speakers will be used.
- Creating a new navigation package will not be done, the existing navigation package will be used.

1.5 Related work

Earlier work on the robot was done in the course SSY226, Design Project in Systems, control and mechatronics in 2023. This course involved several di erent project groups working on di erent projects. One of these projects focused on perception and decision-making for intelligent robots, resulting in the paper "Perception and Decision Making for Intelligent Robots" by Hjertqvist et al [4].

In this project, a map was created using Adaptive Monte Carlo Localization (AMCL) around the area that the robot was to operate in. Additionally, using an o -the-shelf object detection model in the form of basic YOLO, human detection was implemented which in turn allowed for the distance to a human to be obtained. This distance was derived using the depth image that was captured using the RGB-D camera of the robot. On obtaining the depth image, the average depth of all valid registered pixels was calculated. Valid registered pixels represent the objects of interest in the scene and then using the depth image and the object segmentation, the average depth of all valid registered pixels within the object segmentation was calculated. This average depth represented the distance from the robot to whatever object that was detected. Using this, a system allowed the robot to slow down the closer it got to a human and then stop completely when within 2m.

A basic decision tree that illustrates the decisions that the robot takes can be seen in gure 1.4. Building on this foundation that has been laid down, this thesis aims to make decisions depending on various factors and make the robot act accordingly based on the situation it encounters in real-time.

Figure 1.4: The decision tree from the paper "Perception and Decision Making for Intelligent Robots [4]" that describes each of the actions that the TIAGo robot takes depending on various factors.

Integrating an object detection model from Ultralytics YOLO [5] into ROS has been done before by Kimihiro [6]. They use the default YOLOv8 model which is trained on a small subset of the COCO dataset, COCO128, which only contains 128 images from the entire dataset [7] and they speci cally use ultralytics YOLOv8 since it enables exible integration with various robotics applications. This default model that Kimihiro implements into their ROS integration, can recognize some objects to a limited degree but if higher certainty is required, a better trained model needs to

be used. Kimihiro does not investigate any other possible object detection models that could possibly be implemented into ROS but merely implements the ultralytics YOLO model into ROS for ease of use.

One technical paper which was helpful in implementing ChatGPT API was Prog-Prompt: Generating Situated Robot Task Plans using Large Language Models [8] by Jesse Thomason, etc. The goal of this paper was to improve robot task planning by using large language models so that executable action sequences could be generated based on the environment. For this, the authors developed a programmatic prompt framework which included situational environmental awareness, available actions and task planning. This helped the LLM to generate realistic and context-aware actions. The gure 1.5 present in the paper shows how the LLM is prompted to create plans, perform state feedback and carry out actions in a simulated environment using action primitives, available object list and example tasks. This study showed improved task performance and successful deployment on real-time robot. The methods described in this paper like structured prompts with description of the environment, laid the foundation in utilizing ChatGPT API to identify robot's environment and generate relevant responses depending upon particular scenarios.

Figure 1.5: ProgPrompt's work ow for generating and executing plan for robot tasks with large language models (LLMs) like GPT-3 from the paper "ProgPrompt: Generating Situated Robot Task Plans using Large Language Models [8]"

Another technical paper related to ChatGPT and robotics which was relevant to this thesis wasRobotGPT: From ChatGPT to Robot Intelligence[9] written by Hongmei He. The main aim of this paper was to review the principles of ChatGPT and investigate ve di erent types of robot perception which is similar to human senses, including seven types of robot intelligence but excluding intrapersonal intelligence. There is an emphasis on self-awareness, personality and ethics like humans to be considered in robotic systems. Therefore using GPT-3, these seven robotics intelligence are implemented which in turn provides a framework for RobotGPT.

2

Theory

This chapter will present the theoretical concepts and methods that were used for this thesis. Such as how object detection using YOLO works and how ROS can be used for communications between a robot and newly implemented functions. Furthermore, it will also explore how the mapping and localization methods work and how the orientation of a robot can be described using quartenions and Euler angles.

2.1 Object detection using YOLO

Using the RGB-D camera and an image recognition model, it is possible to detect and segment objects in real-time. You Only Look Once (YOLO) is a popular and state of the art object detection algorithm in the eld of computer vision which revolutionized real-time object detection tasks [10][11]. It was also used in the previous work on the robot, as mentioned in the related work section. Unlike other object detection methods, YOLO takes an image as an input and then processes that image through a neural network to detect objects in an image thus, enabling it to make predictions e ciently [12]. As the input image is divided into a grid and predicting bounding boxes and class probabilities, YOLO is very good in terms of accuracy in detecting the desired objects with impressive speed.

YOLO version 8 and the task of Instance Segmentation has been used in this thesis to train a custom dataset [11].

Instance Segmentation is one of the tasks in YOLO that involves precisely identifying individual objects that are present in an image at a pixel level. Unlike the task of object detection which involves only detecting objects and drawing bounding boxes around them, instance segmentation goes a step further by not only identifying individual objects but also segmenting them from the rest of the image, that is, segmenting individual instances of objects in an image at the pixel level [13][14].

Although YOLO is very good at object detection and giving their locations by creating bounding boxes, the problem is that in each grid cell, it treats all instances of the same class as a single entity without di erentiating between individual instances. This is where instance segmentation excels by providing a much better understand-

ing of the scene by not only detecting objects but also segmenting each instance of the object separately. Also, when there is a need to know where the objects are in an image along with their exact shape, instance segmentation is useful. Therefore, the task of instance segmentation is used in training the data so that in an image, it is possible to detect the presence of multiple objects and at the same time, accurately point out the boundaries of each object instance pixel by pixel [13][14].

In order to train the model, the concept of epoch is very important. Epoch refers to one complete pass over the entire dataset during the training process of the machine learning model. Every training example is processed once by the model during each epoch, and its parameters are updated in response to the errors that are observed. The main purpose of training for multiple epochs is to allow the model to gradually learn from the training data so that its performance can improve over the due course of time. The main advantage of exposing the model to the complete dataset multiple times is that it can re ne its parameters and improve its predictive capacities [15].

Therefore, by training a custom dataset using the task of instance segmentation with the help of YOLOv8, the robot can accurately identify individual objects, which is important in navigation scenarios thereby improving its performance in complex and dynamic environments.

2.2 Robot Communication - ROS

ROS (Robot Operating System) is a set of open-source software libraries and tools which is used to develop robotics applications [16]. ROS can be used with several di erent programming languages, however C++ and Python are mainly used for it. Using ROS allows for the subscription and publishing's of topics to di erent nodes, among other things [17].

Nodes are processes that perform computations. A single node is often designed to perform a single task, such as one node performing path planning and another controls a laser range nder. Nodes can communicate with other nodes using the subscription and publishment to topics, or request and response for services. Since the nodes can communicate with each other in di erent ways it allows for a multitude of nodes to interact with each other allowing for complex systems, such as controlling a robot [18].

Topics can be used for exchanging messages between di erent nodes. Nodes can publish a message to a topic and another node can subscribe to that same topic to obtain that message whenever it changes. These messages can include data such as the sensor data from a laser range nder or control commands for moving a vehicle [19]. Figure 2.1 shows a representation of how nodes can subscribe and publish to a topic.

Figure 2.1: A simple representation of how nodes can communicate with each other using topics.

Services enable communication between di erent nodes. A node can send a request to another node and then wait for a response. This makes it suitable for tasks that require con rmation before continuing [20]. Figure 2.2 shows a basic representation of how services works in ROS.

(a) This gure shows a node sending a request message to another node.

(b) This gure shows a node responding to the request message from gure 2.2a by sending a response message.

Figure 2.2: A simple representation of how nodes can communicate with each other using services. The two gures 2.2a and 2.2b show the request and response respectively.

In this project, ROS is critical for the communication between the implemented nodes and the existing software on the robot. It is used for subscribing to important topics from the robot, such as the laser scan data or the TIAGo robots velocity. As well as publishing to topics that the TIAGo robot is subscribing to, such as the navigation velocity.

2.3 Mapping & Localization

With the laser range finder sensor on the TIAGo robot along with odometry data from the wheels, a grid-based occupancy map of the environment that the robot is in can be created using a ready Gmapping ROS package. This is done by using Gmapping which is a SLAM (Simultaneous Localization and Mapping) algorithm that can create grid maps using the data from a laser range finder with the help of a Rao-Blackwellized particle filter [21]. It also uses the odometry data from the wheels to provide an estimate of the robot's pose. A grid-based occupancy map means that the map is divided into discrete cells, forming a grid. Each cell can be one of three different states. Either the cell is occupied by an object, it can be non-occupied and finally it can be non-defined. If a map is created of a completely empty room then each cell representing the room would be non-occupied. The walls of this room would be considered occupied and the cells outside of the walls would be non-defined. One advantage of Gmapping is the ability to create accurate maps in different environments due to its robustness [22].

Obtaining the position and pose of a robot on a completed map can be done by using AMCL (Adaptive Monte Carlo Localization) which is a probabilistic localization algorithm. This algorithm can be used for localizing moving objects on a grid-based occupancy map [23]. This algorithm uses a particle filter along with the real-time data from the laser range finder and odometry data to estimate the location and pose of the robot. This is done by having the robot move around while the AMCL algorithm is running to compare the real-time data to the already created map. As the robot moves, the particles are continuously updated. Initially, the particles are spread all over the map. However, they slowly converge to the most likely poses of the robot based on the comparison between the laser data and the map. Figure 2.3 shows the robot navigating in a corridor where there is a cluster of red arrows in front of the robot which represents the particles which have converged to a similar location.

Figure 2.3: The robot navigating in a corridor where the cluster of red arrows right in front of the robot represent the different particles that have converged. From the paper "Perception and decision making for intelligent robots [4] "

Figure 2.4 shows another TIAGo robot in a warehouse where the localization has just been turned on and the robot has not yet moved [24]. Note that there are red arrows all over the map that represent different possible robot poses. If the robot were to move, the red arrows would start to converge.

Figure 2.4: A TIAGo robot in a warehouse with a multitude of particles that represent possible poses of the robot. Image was originally from the TIAGo tutorial "Localization and path planning" by Jordi Pages [24].

2.4 Quaternions & Euler Angles

The position and orientation of the TIAGo robot consists of the position in x, y and z on the map along with the orientation of the robot in the form of a quaternion. Quaternions are a four parameter representation of the orientation of an object around an axis in 3d-space [25]. A unit quaternion $Q = \{w, \}$ consists of a scalar part and a vector part, w and $= [\ _x;\ _y;\ _z]^T$ respectively. The scalar part, w, is defined as: $w = \cos(\frac{\#}{2})$ and the vector part, as: $= \sin(\frac{\#}{2})\ r$. Where # is the rotation angle and r the rotation axis. This specific way of representing an orientation allows for the description of any orientation at all, which other orientation representations may not allow [25].

Euler angles are a set of three angles that can describe the orientation of an object in 3d-space [25]. In equation 2.1, a representation of those angles can be viewed. They are in the common form of roll-pitch-yaw or ZYX angles. In figure 2.5, the different angles can be seen. The rotation around the red axis shown in figure 2.5 is called roll and is represented by '. Similarly, the rotation around the blue axis also shown in the figure 2.5 is called pitch and is represented by # and the rotation around the green axis, shown in the same figure 2.5 is called yaw which is represented by [25].

$$= [';\ \#;\ \ ]^T \tag{2.1}$$

Figure 2.5: Euler angle representation on top of the TIAGo robot.

Due to the fact that quartenions are not that intuitive to understand [25], the thesis group decided that converting the quartenions to an euler angle would be preferable. To obtain the euler angles from quartenions, some calculations are needed. For this thesis though only one euler angle, the yaw angle is needed so only that calculation will be presented in equation 2.2.

$$= \arctan \frac{2{:}0 \ (w \ z + x \ y)}{1{:}0 \quad 2{:}0 \ (y^2 + z^2)}^{!} \tag{2.2}$$

3

Methods

This chapter presents the details of di erent implementations that were done in this thesis starting with how a map of the surrounding environment was created with the robot, how a custom dataset was created and then trained using YOLO, how the calculation of robot speed, human speed and angle between robot and human was done, the development of an ontology, the di erent nodes that were implemented in python programming language in ROS environment along with a block diagram for the same and how ChatGPT API was implemented for detecting the room. A small pipeline of the di erent sections present in the chapter of methods is shown below in the gure 3.1.

Figure 3.1: Pipeline showing di erent sections present in the chapter of methods

3.1 Mapping with ROS

In accordance to the mapping theory in 2.3, creating a map of the environment that the robot is in can be done by using the Gmapping ROS package [26]. Gmap-

ping works by subscribing to the laser scan message from the robot as well as the odometry from the robot. As was mentioned in the theory, it works by dividing the environment into cells which can be in one of three states, occupied, non-occupied or non-de ned. The Gmapping package assigns a probability of the state of each cell and as it collects more data, it changes this probability for each cell. The localization and mapping by the Gmapping package uses the Rao-Blackwellized particle lter which places a number of particles on the map which represents possible poses of the robot. This lter does not require a map since it performs simultaneous localization and mapping. As the robot is moving around, it uses the laser scan data, a motion model and odometry data to update the probability of these particles. The default parameters of the Gmapping package was used and no other changes were performed when using the Gmapping package.

The key idea of the Rao-Blackwellized particle lter is the estimation of the joint posterior of the map and robot trajectory [27]. Equation 3.1 shows the formula for this calculation. The equation and the following method for the Rao-Blackwellized particle lter that Gmapping uses is based on the paper "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters" by Grisetti et al [21].

$$p(x_{1:t}; m \mid z_{1:t}; u_{1:t}) = p(m \mid x_{1:t}; z_{1:t})\, p(x_{1:t} \mid z_{1:t}; u_{1:t}) \tag{3.1}$$

Where:

- $x_{1:t}$ is the trajectory of the robot.
- m is the map.
- $z_{1:t}$ is the sequence of observations (laser scans).
- $u_{1:t}$ is the sequence of control inputs (odometry).
- $p(m \mid x_{1:t}; z_{1:t})$ is the posterior calculation for the map
- $p(x_{1:t} \mid z_{1:t}; u_{1:t})$ is the posterior calculation for the robot trajectory

Grisetti et al proposes the following method for the calculation of the posterior:

1. Sampling from the proposed distribution means new particles are generated representing possible poses of the robot. Sampling is the process of generating random samples from a proposed distribution in order to investigate the robot pose and map con gurations. The proposed distribution is a Gaussian distribution which is centered around the robot's pose.

   Sampling is done from the proposed distribution by generating an initial guess for the robot's pose from odometry data and re ning this guess using scan-matching. Scan-matching refers to a technique used to compare the sensor data. With this re ning done, a Gaussian proposal distribution is produced

around the most likely pose of the robot.

2. Based on the present map and pose, importance weightings are given to each particle according to the observation likelihood.

3. Resampling is done only if it is needed according to the e ective number of particles N_{eff} , which is calculated as shown below in the equation 3.2:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w^{(i)})^2} \tag{3.2}$$

where: $w^{(i)}$ refers to the weight of i-th particle and N represents the number of samples.

4. Finally, the map estimation is updated based on the trajectory $x_t^{(i)}$ and the latest observations z_t.

After a map has been created and saved, then the localization is done by AMCL instead, as there is no need to continuously create a new map. This also uses a particle lter to determine the position of the robot. The AMCL algorithm places a large number of possible particles on the map that represent the pose of the robot. As the robot moves around, the probability of the particles is updated as more information is obtained from the laser scan data as well as the odometry. The main di erence between the methods is that the AMCL does not simultaneously map and localize itself, AMCL requires a previously saved map to localize the robot.

Creating a map using Gmapping was done by manually navigating around the different rooms, lab, hallway and living room while running the mapping package, the commands for this can be seen in listing 3.1. The rst line turns on the mapping option and the second line starts the navigation package, which is now con gured for mapping.

Listing 3.1: Commands starting the mapping package

```
1 rosservice call /pal_navigation_sm "input: 'MAP'"
2 rosrun rviz rviz -d `rospack find tiago_2dnav`/config/
    rviz/navigation.rviz
```

The Gmapping used its SLAM algorithm which meant that it could create a map and simultaneously localize itself on that map. The manual navigation composed of driving the robot around in the di erent rooms using the joystick controller. As the robot was driven around, it was often stopped and manually performed a complete rotation while stationary to ensure that all walls and objects were detected. When this had been done for all the rooms then map was considered completed by the thesis group and the Gmapping package was turned o and the map was saved on the robot in accordance to listing 3.2. Where the rst line starts the map_server from the Gmapping package which allows for the saving of maps when called for. The second line calls for a ros service which saves the map and a directory can be chosen if desired, however in this listing no directory is chosen.

Listing 3.2: Commands for saving the map

```
rosrun map_server map_saver
rosservice call /pal_map_manager/save_map "directory: ''
  "
```

For this thesis, a map was created, see section 4.2, that contains several di erent rooms, gure 4.2 shows the Lab & adjacent Hallway and gure 4.3 shows the Living room & connecting Hallway.

3.2 Training the YOLO Model

The workings of YOLO was discussed in the section 2.1. In the beginning of trying out the training of YOLO, instead of using o -the-shelf perception system using YOLO which was done by the previous project group, a custom dataset was created which contained images of only the objects that were available in the surroundings of the robot. For detecting doors, door handles were utilized and in the same way for detecting lockers, number locks were used. For door handles and number locks, a custom dataset was created which involved taking photos with an iPhone 13 camera around the surroundings of the robot. The iPhone 13 features a dual 12MP camera system with both wide and ultra-wide options in it [28]. Using the camera, photos of doors and lockers were taken but from those photos, only door handles and number locks were annotated using robo ow the website [29] [30]. Robo ow is a computer vision platform where images can be annotated in a rather simple and convenient way [31]. The annotations for both door handles and number locks can be seen in the gures 3.2 and 3.3 shown below.

Figure 3.2: Door handle annotated using robo ow

Figure 3.3: Number lock annotated using robo ow

In addition to this, it was necessary to include more photos of objects like chairs, laptops, person and tables. Therefore, from the entire COCO dataset that was available on the Microsoft website, only those objects were downloaded and used for training the data, with a total of 1829 images downloaded from the COCO dataset. To further enhance the dataset, information from the robot's camera was used in the form of bag le. A bag le is a type of le format that is used to record and store the data which involves taking videos from the robot's RGB-D camera, in the form of the ROS topic, /xtion/rgb/image. As the robot moved around its surroundings, a video was recorded and then, the video was converted into a list of images. The photos obtained from the bag le of the robot required annotations to be performed on them. Therefore, once again, the Robo ow website[1] was used and annotations were performed on those photos obtained from TIAGo robot [32]. Then, the individual datasets of each objects were combined into one single dataset which consisted of a total of 3216 images present in both the training and validation dataset [33]. The images were split in a ratio of 70:30 between the training and validation sets respectively. Finally, on the combined dataset, data training was performed.

The default parameters used for training the dataset using YOLO are shown in appendix A.1. With exception for the image size and epochs, the rest of the parameters were set as their default values. The data training was performed for 350 epochs with an image size of 640.

Although the image size of 640 is set by default, it was deemed necessary to use

[1] https://roboflow.com/

that same image size since the RGB-D camera that is present in the robot has a resolution supported only for that image size [15].

The YOLO data training was performed two times. Once with a smaller data set and then with a larger dataset.

3.3 Calculating Speed of the Robot and Human

To improve safety, allowing the robot to overtake a human should be done, when the human is not standing right in front of it or moving around. For the robot to comprehend that the human is not standing in its path or moving around the position and subsequent speed of the human needs to be calculated. Calculating the speed of the human requires the distance to the human, obtained by the RGB-D camera, and the angle from the robot to the human.

3.3.1 Calculation of Robot Speed

The speed of the robot is calculated by using the position of the robot with respect to the map. A function is implemented, `robot_speed()`, which is used to calculate the distance travelled by the robot between the current robot position and the previous robot position. This distance is then divided by the time elapsed to obtain the robot's speed. For this to happen, there is an initial check to see if previous positions and time are available or not. If they are available, then the distance covered by the robot is calculated using the Euclidean distance formula which is given below in equation 3.3

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.3}$$

Then, the elapsed time is calculated between the current and previous positions and finally the distance covered by the robot is divided by the time elapsed, leading to the speed of the robot being obtained. To obtain the position of the robot, the topic `robot_pose` is used which contains the position and orientation of the robot based on the AMCL algorithm and the odometry data.

3.3.2 Calculation of angle between Robot and Human

If a human is detected, the angle data is recovered by the detection system representing the horizontal angle of a human with respect to the robot. This angle is useful in indicating where the position of the human is in regard to the robot as well as in regards to the map. This data is then utilized by the robot for further decision-making process. In this case, to adjust its path so that it can overtake a human if not in its collision path.

To obtain the angle from the robot to an object that it sees with its RGB-D camera, first the location of the object on the real-time video feed must be acquired. That is, the pixel coordinates of the object detected within the image. This is done with the YOLO segmentation of the detected object which then gives the center of the object

itself. These pixel coordinates can be used to calculate the horizontal angle between the robot and the object. Note that other pixel coordinates may also work however the center of the object was chosen arbitrarily. Thereafter the center of the video feed itself must also be acquired for the following equations to work. The specifications of the camera is known with its resolution and FOV among other things. Since the resolution is 640x480 pixels then the center coordinates of the video feed is simply (320,240). Using Computer Vision: Algorithms and Applications by Szeliski [34] the focal length, f, can be obtained using the FOV, which is 60° in this case, and center of the video feed, (320,240) with equation 3.4.

$$f = \frac{\text{center of image}}{2 \tan(\frac{FOV°}{2})} \tag{3.4}$$

The focal length is calculated to be $320\sqrt{3}$, the horizontal angle to the object from the point of view of the camera can then be calculated in accordance to equation 3.5 where $_x$ represents the difference between the center of the image and the center of the object [34].

$$_x = \arctan(\frac{x}{320\sqrt{3}}) \tag{3.5}$$

This equation can obtain the horizontal angle to any object from the robot. Furthermore, the pseudo code for this equation that is continuously running as the robot sees objects can be seen in algorithm 1. A top-down view of the map with the robot and an object detected can be seen in figure 3.4 where the angle $_x$ represents the angle between the robot and an object.

Figure 3.4: Calculation of Angle to a green sphere from the RGB-D camera of the robot.

Algorithm 1 Calculate Angle to Image Center

1: Set image_center_x as the width of the image divided by 2
2: Calculate angle_to_center using the arctangent function:
3: Subtract the x-coordinate of the object's center from image_center_x
4: Divide by a constant value (e.g., $320\sqrt{3}$) representing the focal length
5: Convert angle_to_center to degrees
6: Assign angle_to_center_degrees to angle_to_center
7: Return angle_to_center

3.3.3 Calculation of Human Speed

A human's speed is calculated using the depth part present on the robot's camera. Note that only the speed of the human is calculated as the direction that the human is moving in is not considered. There is a function which is implemented called human_speed() which is used to detect the presence of a person and if detected, calculates the speed of that person. For checking if a person is detected or not, first a regular check is performed every 1 second over the detected objects to check if the object detected is a human or not. If a human is detected, the position of the human based on the global map is saved using the angle equation 3.5 and the distance. Then at the next time step, the euclidean distance, in accordance to equation 3.3, between the current and new position of the human is done to obtain the speed that the human is moving at. In algorithm 2, part of the code can be seen.

Algorithm 2 Human Speed and Position Calculation

```
procedure calculate_human_speed :
    Initialize current_human_position, current_time, and speed to None or 0.0
    for each detection in self:detections do:
        if detection is human and confidence_score > 0.8 then :
            current_human_position CalculateHumanPosition (detection:depth, detection:angle_to_center)
            current_time current system time
            speed calculate speed based on position change over time
            Publish speed using self:human_speed_pub
            Update prev_human_position and prev_time
            break
        end if
    end for
end procedure
procedure calculate_human_position (depth, angle_to_center):
    Convert angle_to_center to radians
    Align angle_to_center with robot's yaw angle
    Calculate x and y position in robot's frame using depth and angle_to_center
    Transform x and y to map coordinate frame using robot's position
    return x and y in map frame
end procedure
```

3.4 Ontology-based reasoning

Protege[2] is a free, open-source ontology editor developed by Stanford University which is useful and popular for creating new ontologies[35][36][37][38]. Creating an ontology can be useful for categorizing classes of different objects, and making the classes have properties that are inherent to any created instance of a specific class [39]. A class is a category or group of objects that share common properties. Such as the category of Furniture which contains the subclasses of "Benches", "Tables" and "Chairs" in this ontology.

[2] https://protege.stanford.edu/about.php

Figure 3.5: The ontology created with the di erent objects created as classes.

The picture of the ontology that was created for this project using protege software is shown below in gure 3.5. The ontology was created with the purpose of dividing the di erent objects that could be found in the robot's vicinity into di erent classes. Note that for example "Benches" belong to the class "Furniture" which in turn belongs to the class "Objects". If a new class is to be included then its placement would depend on its relationship with the existing classes. For example, if a the new class is "Cupboard", it would be placed under the class of "InteractableObjects", similar to the "Locker". Having this ontology is important because it provides a structured way to manage object information, enhancing the robot's ability to locate objects within its environment. This structured knowledge representation is important for tasks such as the object recognition, which will be further explored in section 3.5.

3.5 Implementing ROS Nodes

The diagram displayed below in gure 3.6 shows a system of ROS nodes that are used, represented by blue circles and the connections to other nodes represented by green arrows labelled with the corresponding topics. The direction of the green arrow represents whether or not a node is publishing to a topic or subscribing. For example, the Wall Detection node subscribes to the /map and /laser topics and publishes to the topic, /wall_distance, which the Decision-Making node is in turn subscribing to. The text connected to the green arrows that aren't pre xed with a forward slash merely represent an input of some miscellaneous kind, such as the response from ChatGPT or the YOLO model input. Each input will be further explored in the explanation for each node. This interconnected system of ROS nodes shown in gure 3.6 work together in supporting the robot's autonomous navigation, object detection, wall detection, human-robot interaction and decision-making processes.

Figure 3.6: Block Diagram of ROS nodes used in the thesis. Blue circles represent nodes, and green arrows show topic communications. For example, the Wall Detection node subscribes to the topics /map and /laser, then publishes to the topic /wall_distance, which is subscribed to by the Decision-Making node. Text with no forward slash represents inputs that aren't topics, such as the response from ChatGPT.

Decision-Making is the main node which uses the information from the other nodes to help robot take the appropriate actions and decisions. This main node receives navigation velocity topic, /nav_vel from the navigation node, wall distance topic, /wall_distance from the wall detection node, human speed topic, /human_speed from the human speed node and current room topic, /current_room from creating instance node. This node also takes input from ChatGPT API so that it can interact with humans by speaking out its intentions which is given as answer from ChatGPT based on the situation it encounters. Using these topics as input information, the main node gives output to /Robot_vel topic using command velocity topic, /cmd_vel which is used in controlling the movements of the robot. The other output given by the main node is text-to-speech goals, /tts/goal which is used as a speaking functionality so that the robot can speak whenever required. Below the detailed information about the topics it is subscribing to as well as the topics it publishes to can be viewed:

- Inputs to Decision-Making Node:

  Navigation Velocity Topic (/nav_vel) : from Navigation Node

  Wall Distance Topic (/wall_distance) : from Wall Detection Node

  Human Speed Topic (/human_speed) : from Human Speed Node

  Current Room Topic (/current_room) : from Creating Instance Node

ChatGPT API Input : For the creation of what the robot should say.

- Outputs from Decision-Making Node:

  Command Velocity Topic (/cmd_vel) : to Robot Velocity Controller

  Text-to-Speech Goals (/tts/goal) : to Text-to-Speech System

The object detection node uses a machine learning model YOLO, which involves training a custom set of data, along with image, /image and depth image, /depth_image to help the robot in identifying di erent objects and their location that are available in its surroundings. The result from this node is given to the node called Creating Instance and Human Speed in the form of the topic /det_res or detection result, which contains which class the object detected belongs as well as distance and angle from the robot to the object. Below, a brief explanation of the topics it uses can be seen.

- Inputs to Object Detection Node:

  Image Topic (/image) : from Camera Node

  Depth Image Topic (/depth_image) : from Depth Sensor Node

  YOLO : from the YOLO model

- Outputs from Object Detection Node:

  Detection Result Topic (/det_res) : to Creating Instance and Human Speed Nodes

  Purpose : Provides information about detected objects including class, distance, and angle from the robot to the object.

The node called Creating Instance uses Prolog, which is a logic programming language to create instances belonging to the class of whatever object was detected, along with their position on the map. Furthermore, Prolog is used for querying which instances are close to the robot and creates a list of the nearby objects. This list is then used by ChatGPT along with the prede ned query to obtain the room that the robot is in. The output of this node is published to the topic /current_room which the Decision-Making node is subscribed to and it is simply the room that the robot is in. The topics that are subscribed to and published to can be seen below.

- Inputs to Creating Instance Node:

  Detection Result Topic (/det_res) : from Object Detection Node

  Purpose : Provides information about detected objects including class, distance, and angle from the robot.

- Outputs from Creating Instance Node:

  Current Room Topic (/current_room) : to Decision-Making Node

  Purpose : Indicates the room that the robot is currently in.

The human speed node uses the robot's pose using the topic /robot_pose along with the detection result, or det_res, so that it can calculate the speed of a human moving around. This speed is published to the topic /human_speed. The topics related to the human speed node can also be seen below.

- Inputs to Human Speed Node:

  Robot Pose Topic (/robot_pose) : from Localization Node

  Detection Result Topic (/det_res) : from Object Detection Node

- Outputs from Human Speed Node:

  Human Speed Topic (/human_speed) : to Decision-Making Node

  Purpose : Provides information about the speed of humans to the robot.

Wall Detection node uses map and lidar, /map and /lidar respectively, to detect walls. It provides wall distance, /wall_distance to the decision-making node so that the robot can get to know if it is closer to the wall or not so that it can take appropriate actions.

- Inputs to Wall Detection Node:

  Map Topic (/map) : from Mapping Node

  Lidar Topic (/lidar) : from Lidar Sensor Node

- Outputs from Wall Detection Node:

  Wall Distance Topic (/wall_distance) : to Decision-Making Node

  Purpose : Provides information about the distance to walls, enabling the robot to take appropriate actions based on proximity to walls.

The Speaking node receives text-to-speech goals, /tts/goal topic from the decision-making node, which enables the robot to speak out verbally when it encounters humans within a particular distance and also depending upon the location of the room it is currently located.

- Inputs to Speaking Node:

  Text-to-Speech Goals Topic (/tts/goal) : from Decision-Making Node

Purpose : Enables the robot to speak verbally based on specific criteria such as human proximity and current room location.

- Outputs from speaking Node:

  Audio : In the form of speech from the speakers.

3.5.1 Creating instances of objects and room detection Node

The implementation of this part involves enabling the robot to create instances of objects within its environment when detected with its camera and determine in which room the robot is located. Instances in this case are specific entities created that belong to their class, such as chair or bench. Furthermore, these instances contain an associated position. In summary, the instances are specific entities created which belong to a specific class and they also contain a position.

The image dataset which was trained using YOLO detection algorithm is used by the robot to detect different objects in its environment and processes the object detection results which is received through the detection result topic, det_res. When such object detection messages are received, the robot examines these detected objects by using the confidence scores which is how confident the YOLO model is that the object belongs to a specific class. This confidence score is used to filter out classes that the model is uncertain about. Therefore using an arbitrarily chosen threshold of 80% to determine whether or not an instance should be created. If the confidence score is lower than 80% then an instance of the specific object is not created. Furthermore, since a human can very easily move around, creating an instance of a human is not done as the position belonging to such an instance would be inaccurate as soon as the human would move.

Using the depth and angle information, the position relative to the current pose of the robot is calculated for each valid object. The angle to the object is calculated in exactly the same way as for a human, as was explained in 3.3.2. Furthermore, the yaw angle of the robot which is also obtained and calculated from the robot quaternions, see 2.4 in the theory.

Using this depth and angle information, the function creating_instance can determine the position of each detected object with respect to the current robot's position and orientation. Based on the depth and angle information of the detected objects, the x and y coordinates of the detected object relative to the robot's orientation is calculated using trigonometric calculations which can be seen in the algorithm 3, lines 4 to 16. These computed x and y coordinates represent the position of an object relative to the robot. Then, the computed x and y coordinates are added to the robot's current position in order to transform them into absolute map coordinates. This transformation is done to determine the location of the object within the global map. In summary, aligning the angle to the object and the yaw angle of the robot as well as using the distance to the object, the location of a detected object can be placed correctly on the map itself.

When a new instance belonging to a speci c class is to be created, the function checks all instances belonging to that class and compares their positions to the position of the new instance. The function calculates the euclidean distance between the position of the detected object and the previous instances of the same class using the formula that is speci ed in the equation 3.3. If this distance is less than a minimum threshold of 2 meters to another object of the same class, then the object detected is considered to be too close to an existing instance of the same class and another instance is not created. Therefore, creating instances of objects is dependent upon the prede ned minimum distance threshold which ensures that new instances are not created if they are very close to the existing instances of the same class. This prede ned threshold was chosen arbitrarily.

Furthermore, in order to only create instances of objects in valid positions on the map, not outside of the known rooms. This is achieved by not creating an instance before the validity of the position is checked. This is done by comparing the position of a possible object location to the same coordinates in the occupancy grid map. In algorithm 3, the pseudo code for creating instances is shown.

Algorithm 3 Create Instance

```
function CreateInstance
    for each detected object do
        if object is not "Person" and confidence score > 0.80 then
            Calculate object position in map coordinates:
                Convert angle to radians and align angle with robot's yaw:
                angle_object_rad = math.radians(angles[i])
                aligned_angle = robot_yaw - angle_object_rad
                Calculate object position relative to robot:
                  object_x = depths[i] * math.cos(aligned_angle)
                  object_y = depths[i] * math.sin(aligned_angle)
            if object position is NaN then
                continue                    . Skip processing if position is NaN
            end if
                  robot_x, robot_y = robot_pos
                  object_map_x = robot_x + object_x
                  object_map_y = robot_y + object_y
            if object position is valid then
                Initialize too_close as false
                for each existing instance of the same class do
                    Calculate distance to existing instance
                    if distance < min_distance_threshold then
                        Set too_close to true
                        break
                    end if
                end for
                if not too_close then
                    Check map occupancy at object position
                    if position is free then
                        Create new instance in the knowledge base
                        if instance creation is successful then
                            Increment instance counter
                            Visualize object in Rviz
                        else
                            Log warning for failed instance creation
                        end if
                    end if
                end if
            end if
        end if
    end for
end function
```

Moreover, to make things more interactive, the implementation includes the use of ChatGPT API from OpenAI to recognize which room the robot might be in. Based

on the detected objects in its surrounding environment, the robot uses this API to recognize the room it is currently present in. This room detection is based on the physical closeness of the detected objects in the surroundings of the robot. By analyzing and detecting speci c object combinations nearby, the room where the robot is present can be inferred. In order to do this, a structured message that includes a list of the detected objects and their classes is sent to the ChatGPT API. Note that the two types of queries that were sent to ChatGPT can be viewed in the results 4.5.1.1. Then, based on that information, API processes it and replies with the most likely room based on the objects that are present. The use of prolog helps facilitate knowledge representation and reasoning, with newly created instance positions asserted into the knowledge base. In algorithm 4 the pseudo code for obtaining the room can be seen.

Algorithm 4 Get Robot Room

```
function GetRobotRoom
    Initialize nearby_objects as an empty list
    for each class name and instance id in instance_counts do
        for each instance id do
            Query to get instance position
            for each solution in query results do
                Calculate distance from robot to instance
                if distance < room_distance then
                    Add class name to nearby_objects
                end if
            end for
        end for
    end for
    Count occurrences of each object class in nearby_objects
    Format nearby_objects as a string
    if nearby_objects in room_cache then
        Retrieve room from cache
    else
        if nearby_objects is not empty then
            Create prompt for OpenAI API
            Get room response from OpenAI API
            Cache the response
        else
            Set room to "Unknown"
            Log "No objects detected"
        end if
    end if
    Publish room name
    return room, nearby_objects
end function
```

3.5.2 Developing the Wall Detection Node

The wall detection node has been implemented so that the robot can detect walls and objects that are present within its vicinity. For this implementation the occupancy grid map '/map', robot's position '/robot_pose' and LIDAR data '/scan' are the topics that are used.

There is a function, detect_walls which uses the occupancy grid map to find the cells that indicate wall which are less than 0.5m from the robot. The 0.5 m is primarily so it doesn't calculate the distance to walls far away unnecessarily. A wall is detected if there are more than 9 points, as detected by the laser range finder, in a straight line. If a wall is detected then the distance to the nearest wall is published. The algorithm for this function can be viewed in algorithm 5 which also uses another function to calculate if the wall is flat or not which can be seen in algorithm 6.

Algorithm 5 Detect Walls

```
1: function DetectWalls
2:     if map_data is unavailable or robot_pose_data is unavailable then
3:         return
4:     end if
5:     Initialize closest_wall_distance as -1
6:     for each cell (x, y) in the occupancy grid do
7:         if cell value at (x, y) is 100 then
8:             Calculate distance to wall from robot's position
9:             if distance is within 0.5 meters and wall is flat then
10:                Update closest_wall_distance if current wall is closer
11:            end if
12:        end if
13:    end for
14:    Publish closest_wall_distance or -1 if no wall is close enough
15: end function
```

Algorithm 6 Is Flat Wall

```
1: function IsFlatWall (x, y, map_width, map_height)
2:     for each cell (dx, dy) in a 3 × 3 neighborhood around (x, y) do
3:         if cell value at (x + dx, y + dy) is not 100 then
4:             return False
5:         end if
6:     end for
7:     return True
8: end function
```

The detect_objects_in_front function is used to analyze the laser range finder data for detecting objects in a 90-degree slice in front of the robot. It calculates how many points are present within 2m of the robot. This can be used specifically

for low elevated objects that cannot be seen by the camera as the objects are too close to the robot and too low to be viewed by the camera. Thereby, using the laser data, objects can be detected that the camera cannot see, which is used to increase safety. In algorithm 7, the calculation for this can be found. Note that it also sends the number of points within 2m, since the laser may return erroneously points which aren't real so, using the number of points within a distance can be used to increase robustness. The 2m distance is used so that unnecessary computations aren't performed as there is no need to calculate if any object is that far away since the camera would see it. This result is used in the decision-making node, if an object is detected close to the robot then the decision-making node can make the decision to slow down regardless of which room the robot is in.

Algorithm 7 Detect Objects in Front

```
function DetectObjectsInFront (scan_data)
    if scan_data is unavailable then
        return
    end if
    Extract ranges from scan_data.ranges
    Calculate total number of measurements num_ranges = len(ranges)
    Calculate angular resolution (radians per measurement):
        angular_resolution_radians =  / num_ranges
    Calculate number of measurements within the 90-degree slice in front of the robot:
        num_meas_in_slice = int((  /2) /angular_resolution_radians)
    Calculate start and end indices for 90-degree slice:
        start_index = int((num_ranges - num_meas_in_slice) / 2)
        end_index = start_index + num_meas_in_slice
    Count ranges less than 2 meters within the slice:
        object_count = sum(1 for i in range(start_index, end_index + 1) if ranges[i] < 2)
    Find minimum distance in the slice:
        object_distance = min(ranges[start_index:end_index + 1])
    Create and publish message with object distance and count
end function
```

3.5.3 Testing the Decision-Making Node

The decision-making node is the main node that is used in this thesis which takes the information from all the other nodes described above in the sections 3.3.1, 3.3.3, 3.5.1 and 3.5.2 and helps in the decision-making process of the robot depending upon the scenario it encounters.

Various ROS topics such as detection results, wall distance, object distance, human speed, robot speed and current room are subscribed to by this node. Publishers such as decision topic and tts (text to speech) goal are used to send the decision and text-to-speech messages. In addition to this, OpenAI API is also used in this

node so as to generate speci c speech responses for the robot so that it can speak out the answers given by ChatGPT depending upon the location and situation it encounters. Along with this, the speed of the robot is also plotted simultaneously to monitor the performance and speed of the robot.

There is a mainpublish_ decision function which is used to help the robot in taking the proper decision depending upon the situation it encounters by adjusting the robot speed as well based on the location or the surrounding environment it is present in. Here are some of the scenarios which have been programmed in the robot:

1. If the location detected by the robot isLab :
    - It will never go faster than 50 % of the originally requested speed
    - if it sees a human within a distance of 2m, it will stop and at the same time, speak out the answer that is given by ChatGPT.
2. If the location detected by the robot isLiving Room :
    - if it detects a wall in less than 0.4m, it will slow down its movement speed and at the same time, speak out the answer that is given by ChatGPT.
    - if it sees a human within a distance of 1m, then it will completely stop and at the same time, speak out the answer that is given by ChatGPT.
    - if it sees a human moving at fast speed which is greater than 0.4, it will stop completely and at the same time, speak out the answer that is given by ChatGPT.
3. If the location detected by the robot isHallway :
    - if it sees a human within a distance of 1m and within a collision path it should stop completely and at the same time, speak out the answer given by ChatGPT.
    - if it sees a human moving at a speed greater than 0.2, then it should slow down its movement speed and at the same time, speak out the answer given by ChatGPT.

Therefore, the main goal of this decision-making node is to ensure safe autonomous navigation of the robot around its surrounding environment, keeping in mind to take appropriate actions when encountering humans depending on speci c locations it is present in. These scenarios are tested and the outcome of the tests can be seen in the results section.

The pseudo code for the di erent decisions it makes when in di erent rooms can be viewed in the appendix A.2 to A.4. However in algorithm 8 the pseudo code for the publish decision function can be seen. Note that it only checks the decisions for

speci c rooms when it is not speaking. As can be seen in line 12, this is to make sure it continues doing what it did when it started speaking.

Algorithm 8 Publish Decision

```
1: Input: classes, distances
2: if not joystick_priority then
3:     setting default values to variables
4:     for each object in detected_objects do
5:         if object is human then
6:             if detected_distance < distance then
7:                 distance ← detected_distance
8:                 angle ← detected_angle
9:             end if
10:        end if
11:    end for
12:    if speaking then
13:        if remember_to_stop_movement then
14:            stop_movement ← true
15:            skip_adjust_speed ← true
16:        else if remember_half_movement then
17:            target_speed_ratio ← 0.5
18:        end if
19:    else if object_distance < 0.1 and object_count > 10 then
20:        target_speed_ratio ← 0.25
21:        skip_adjust_speed ← true
22:    else
23:        if current_room == "Lab" then
24:            target_speed_ratio ← lab_decision(distance, angle, index)
25:        else if current_room == "Living_Room" then
26:            target_speed_ratio ← living_room_decision(distance, angle, index)
27:        else if current_room == "Hallway" then
28:            target_speed_ratio ← hallway_decision(distance, angle, index)
29:        end if
30:    end if
31:    if not skip_adjust_speed then
32:        Set new speed using target_speed_ratio
33:    else if stop_movement then
34:        Set speed to 0
35:    end if
36:    publish_decision_message(decision_message)
37: end if
```

3.5.4 ChatGPT API Implementation

ChatGPT is a large language model which uses natural language processing so that human-like responses are generated. Ask questions to ChatGPT and it will respond

similar to a human [40]. Large Language Models (LLM) are a type of AI program, trained on very large sets of data that are capable of recognizing and producing human-like texts. They are developed as a result of machine learning algorithm and they have been given the name as large because of being trained on an incredibly large sets of data. This dataset could be a list of websites or books and since being trained such an enormous dataset, this enables the LLMs to learn from that data that are fed to it [41] [42].

ChatGPT API which stands for ChatGPT Application Programming Interface is a tool developed with an aim to integrate the abilities of natural language model such as ChatGPT into any application the user wants to use with, such as python code. For this, the most important thing is to use an API key that OpenAI provides in its account to access the API. Also it is very important to note that rate restrictions are applicable for the usage of API, that is, some amount of money is needed in the OpenAI account for API to work as intended in the application where it should be used [43] [44].

Hence, after implementing ChatGPT API, results were obtained as answers from ChatGPT when a few detected objects were sent to it and queried by asking for the room. The main purpose is to check how accurate and precise ChatGPT is by answering the room when queried with the same set of objects multiple times. This was done by generating various object combinations and querying to ChatGPT for each object combination as to what room are those objects are located in.

The process of giving ChatGPT a list of objects and querying it for the room it is present in was done twice in order to compare the two results that were obtained. Both times, 41 di erent object combinations were given to ChatGPT with each object combination given to it 10 times to check if ChatGPT is giving the same answer about the location of those particular objects in the respective room.

A total of 410 object combinations were tested for each query. The rst time this process was done by querying a list of all the object combinations generated with vague queries and observing the answer given by ChatGPT as to what room are those objects present in. The second time, same process was repeated but this time with a bit more of precise and clear query given to ChatGPT.

4

Results

This chapter presents the different results obtained related to this thesis starting with testing out the possible sensors, analyzing the different maps created in the lab, hallway and living room, evaluating and discussing about the confusion matrix obtained after training the dataset using YOLO along with some limitations observed, evaluating the instance creation, the room detection using the ChatGPT API and wall detection. At last, analyzing the navigation of the robot in different rooms given different scenarios.

4.1 Identifying Possible Sensors for ROS Integration

The RGB-D camera as well as the laser rangefinder are already essential for any mapping or object recognition to be done. However, the sonar on the back of the robot needs to be investigated. This was done by starting the program RVIZ to visualize the sonar data and testing moving objects and people in front of the sonar. RVIZ is a 3D visualizer for the Robot Operating System (ROS) framework [45] which can be used for visualizing the output from different sensors on the robot. When testing the sonar in RVIZ, it could be seen that it shows the distance detected with its three sonars on the backside of the robot. This can be seen in figure 4.1 where a person is standing right behind the robot and the center cone is very much smaller than the ones next to it. Furthermore, the sonar on the robot is only recommended to be used within 1 m [3].

Hence, the limitations of the sonar to 1m as well as the fact that it only shows object behind it makes it hard to find any meaningful use with the sonars. Because of this, the sonars were disregarded for the purposes of this thesis.

Figure 4.1: The visualization of the sonar on the robot when detecting a human.

4.2 Analyzing Mapping Results

Figure 4.2 shows the occupancy map of the lab, as well as the adjacent hallway. Note the gray color which represents open non-occupied positions. The color that resembles light green represents either occupied locations or unde ned locations. This color can be observed in the gure either circled in red or pointed at by the yellow arrow, which describes occupied or unde ned positions respectively. Figure 4.3 also shows the map of the living room, as well as the adjacent hallway.

Figure 4.2: Map of the lab as well as the surrounding hallway

Figure 4.3: Map of the living room and connecting hallway.

Figure 4.4 shows the map of the lab, as well as the adjacent hallway. However, note the red and yellow circles which illustrate errors in the map creation, inside those circles there are very small spots that resemble light green, which represents occupied positions. The small light green spots are not easily discernible however the accompanying red and yellow circles should highlight that they are there. The red circle means that there was an object there while mapping, while the yellow circle means that there wasn't an object there in reality, however according to the map there was one.

Figure 4.4: Map of the lab as well as the surrounding hallway with actual objects circled in red and objects that do not exist in yellow

As could be seen in the gures 4.2, 4.3 and most speci cally in gure 4.4, there were

objects seen on the map that actually weren't there. This creates a hollow part of the map that is treated as if it was outside of the map. This is a limitation of the laser range nder, it returns erroneous data at some points. However, as can be seen in the gures, the non existent objects or hollow areas are not large on the map and therefore the navigation can often navigate around the objects at no great cost.

4.3 Evaluating the YOLO Model

In gure 4.5, a simple demonstration of the YOLO model detecting objects in the lab can be seen. Note that the con dence score is shown next to the segmentation of the object, which shows how con dent the model is in the classi cation of a speci c object.

(a) This picture shows several objects being recognized by the YOLO model along with the con dence score for each object.

(b) This shows the YOLO model recognizing several di erent objects as well as a human in the lab, all with a con dence score.

Figure 4.5: The two gures show (a) the YOLO model recognizing objects and (b) the same view but with a human as well.

At the end of data training, a normalized confusion matrix was obtained which is very useful in visualizing the performance of the trained model. A confusion matrix is a table that is used to show how many times the model correctly and incorrectly classi ed an object using diagonal and o -diagonal elements respectively [46][47].

The YOLO data training was performed two times and the performance of the rst dataset was not that great since the training ended at 298th out of the full 350 epochs since the mAP-50 value stopped increasing after it reached 0.69. mAP-50 is an important performance parameter used for measuring the accuracy of the model. In the gure 4.6, the resulting confusion matrix obtained from the YOLO model that was trained for the rst time can be seen.

Figure 4.6: Normalized Confusion Matrix obtained after the rst training

In an e ort to obtain a better result in terms of accuracy of the model, the training was repeated once again for the second time with a larger dataset and the output was better with the entire training getting completed with a value of mAP-50 at 0.79. In the gure 4.7, the resulting confusion matrix obtained from the YOLO model that was trained for the second time can be seen. The normalized gure has the sum of 1.0 for each row, which represents the proportion of predictions for each class.

Figure 4.7: Normalized Confusion Matrix obtained after the second training

In the confusion matrices shown in gures 4.6 and 4.7, the x-axis represents the actual classes present in the dataset and the y-axis represents the model's prediction of the dataset. For each class, the diagonal elements represent how frequently each class was accurately identi ed by the model. For example, on carefully observing the normalized confusion matrix in gure 4.7, 84% (0.84) of the laptops were correctly identi ed by the model as laptops and in the same way, 94% of the benches were accurately identi ed by the model as benches. The o -diagonal elements show the objects identi ed incorrectly. For example, in the gure 4.7 of normalized confusion matrix, 12% of number locks were incorrectly identi ed as background. Therefore, the higher the value of diagonal elements gives an indication of a good and accurate model.

By comparing the two confusion matrices obtained at the end of the training in gures 4.6 and 4.7 respectively, it is possible to observe that the model prediction for door handles increased from 76% to 85%. There was a slight increase in the prediction of number lock from 90% to 91% with bench remaining consistent at 94%. There was a signi cant improvement in the model prediction for laptop from 74% to 84% along with an increase in accuracy for chairs from 33% to 39%. The class of person saw a slight increase in prediction from 62% to 63% with a minor decline observed in the class of tables from 88% to 84%.

An important observation in the gures 4.6 and 4.7 is the presence of another class named 'background'. Background was not included in the custom dataset that was created to train the data. Background refers to a class of objects present in an image that are not benches, chairs, person, laptop, number locks, door handles and tables which it was trained to recognize. This background class then represents any part of the image which doesn't contain any of the classes that it should train on. This can be useful in interpreting any false positives or false negatives in the detection of the di erent objects.

Another important observation to be noted is that there was a reduction in misclassi cations as well. Misclassi cation of chair as background decreased from 67% to 60% and the same was with door handles with a reduction from 22% to 13%. The misclassi cation of laptop as background also reduced from 26% to 16%. Overall, the normalized confusion matrix obtained after the second training can be considered as a good improvement over the one obtained after the rst training because of better accuracy and performance of the model along with some reduction in the misclassi cation rate in the background class.

Since the task of annotating was done by the group, it could be possible that the annotations were not done with complete accuracy for some of the objects. It is possible very small but important portions of an object may not have been annotated correctly and this could have led to the training incorrectly deducing that an object is part of the background.