
Context-Aware navigation and Semantic Perception for Intra-factory Logistics

Master Thesis
Group 1061

Aalborg University
Robotics



Robotics
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Context-Aware navigation and Semantic Perception for Intra-factory Logistics

Theme:

Master Thesis

Project Period:

Spring Semester 2025

Project Group:

1061

Participant(s):

Rasmus Lilholt Jacobsen
Wallat Bilal

Supervisor(s):

Dimitirs Chrysostomou

Page Numbers: 67**Date of Completion:**

June 4, 2025

Abstract:

This project explores how a service robot can perform context-aware navigation based on human-given commands. The system combines semantic mapping, object detection, natural language processing, and a navigation stack to allow the robot to interpret and act on voice or text instructions. The robot builds a semantic map by identifying and clustering objects, and then infers room labels using an ontology graph. Commands are processed through an NLP module that extracts relevant keywords and maps them to locations using the semantic map and ontology. The robot can then navigate to the requested location and detect objects in the area. The entire system is tested in a simulated environment using ROS2 and NAV2. While there are some limitations, such as the need for the robot to stop to record object poses and the use of center-point navigation targets, the results show that the system can handle mid-level context-aware tasks. Future improvements could include more advanced scene understanding, dynamic goal selection, and object search behaviors.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

1	Introduction	1
1.1	Initial Problem statement	1
2	Problem Analysis	2
2.1	Motivation	2
2.1.1	Service robots	2
2.2	Semantic mapping	6
2.2.1	Pipeline	9
2.2.2	Image segmentation	10
2.2.3	State-of-the-art CNN	12
2.3	Understanding humans	14
2.3.1	Keyword Extraction	15
2.3.2	Ontology	16
2.3.3	Navigation	17
2.4	Problem / Use case	17
3	Problem Formulation	20
3.1	Final Problem State	20
3.2	Objectives	20
3.3	Delimitations	21
4	Design	22
4.1	Simulation Environment	22
4.2	System overview	24
4.2.1	Ontology	24
4.2.2	Natural Language Processing Node	26
4.2.3	Object Detection Node	26
4.2.4	Semantic mapping node	27
4.3	ROS Node Graph	29
4.3.1	Semantic mapping overview	30
4.3.2	Context aware navigation overview	31
5	Implementation	32
5.1	Semantic mapping	32
5.1.1	Object Detection output processing	33
5.1.2	Clustering objects	36
5.1.3	Location area estimation	41
5.1.4	Location inference	42

5.2	Context aware navigation	44
5.2.1	NLP node	45
5.2.2	Navigation	47
6	Test setups and results	48
6.1	Test Setups	48
6.1.1	Test 1: Object Detection	48
6.1.2	Test 2: Keyword Extraction	50
6.1.3	Test 3: Location Inference	51
6.1.4	Test 4: Semantic mapping	53
6.1.5	Test 5: Command execution	53
6.2	Test Results	55
6.2.1	Test 1: Object Detection	55
6.2.2	Test 2: Keyword Extraction	56
6.2.3	Test 3: Location Inference	56
6.2.4	Test 4: Semantic Mapping	57
6.2.5	Test 5: Command Execution	58
7	Discussion	59
7.1	Test results	59
7.2	Future work	61
8	Conclusion	63
	Bibliography	64
A	Appendix	68
A.1	panoptic segmentation node	70
A.2	Simple dbscan	72

Preface

Aalborg University June 4, 2025

This report is a master thesis from Aalborg University in the field of Robotics. The project was completed by the group as part of the master's program and focuses on developing a context-aware navigation system for a service robot using semantic mapping and natural language processing.

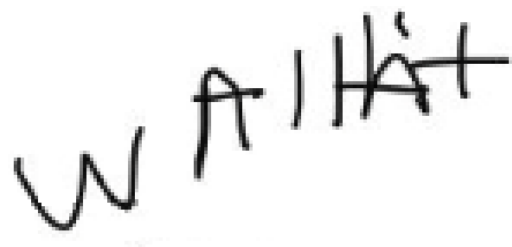
The project was a great opportunity to explore how robots can understand and respond to human commands in dynamic environments. Through this work, we gained a deeper understanding of semantic perception, navigation, and integration of different robotic components in ROS2.

We would like to thank our supervisor, Dimitris Chrysostomou, for his guidance and helpful feedback throughout the project.

The codebase for the project can be found on [Github](#) and videos of the testing can be found at this [link](#).

Rasmus Lilholt Jacobsen

Rasmus Lilholt Jacobsen
rlja20@student.aau.dk

A handwritten signature in black ink, appearing to read 'WALLAT' in a stylized, cursive font.

Wallat Bilal
<wbilal18@student.aau.dk>

Acronyms

BERT Bidirectional Encoder Representation from Transformers. 16, 50

CNN Convolutional Neural Network. ii, 7, 8, 10, 12, 13, 26

DBSCAN Density Based Spatial Clustering of Applications with Noise. 28, 38, 41

DOF Degrees of Freedom. 3, 4

FCN Fully Convolutional Network. 10

GRU Gated Recurrent Unit. 16

HRC Human Robot Collaboration. 1

HRI Human-robot Interaction. 2

IoU intersection over union. 26

LLM Large Language Model. 16, 25, 26

LSTM Long Short-Term Memory. 15, 16

MBR Minimum Bounding Rectangle. 28–30, 41, 62

NAV2 Navigation2. 6, 9, 10, 17, 44, 45, 47, 53

NLP Natural Language Processing. 1, 10, 15–17, 31, 44, 45, 47, 50, 59–61

OWL Web Ontology Language. 16, 24, 42

RNN Recurrent Neural Network. 15, 16

ROS Robot Operating System. 6, 17

SLAM Simultaneous Localization and Mapping. 9, 17, 60

1 Introduction

Over time more and more robots are being implemented in different environments. Robots are being used as tour guides, elderly help and other tasks which all fall under the concept of a robotic assistant. Moving to Industry 5.0 introduces the concept of Human Robot Collaboration (HRC) where collaborative robots being introduced into factories where the operators are able to instruct robots on specific tasks. It is not exclusively manipulators which are becoming collaborative, mobile robots are also being introduced and working along people in factory settings. Static maps of the factories makes it easy for the mobile robots to go from point A to point B however, the problem is not so clear cut since the environment is much more dynamic and people are constantly working around the robot. In an environment such as that simply taking the shortest path could easily lead to accidents or disrupting the people working around it. To tackle this problem the robot would need to be able to collect data from the environment upon which a context can be constructed.

As a robot assistant, how the user interacts with the robot is an important aspect. There are a few possibilities for how the interaction could be done, but the most natural for humans by far is through speech. This project focuses on building a system that combines semantic mapping, Natural Language Processing (NLP), and context-aware navigation for a mobile service robot. The goal is for the robot to take in a command from a user, process the intent using NLP, use semantic information about its environment to interpret the command, and perform the requested navigation task.

1.1 Initial Problem statement

"How can a personal service robot understand a human given command to perform navigation tasks."

2 Problem Analysis

This chapter will be focusing on the motivation behind context aware navigation, this includes looking at the motivation for service robots in general.

2.1 Motivation

Industry 4.0 was introduced in 2011 by the European union where the idea of having fully automated factories where robots handled as much of a assembly line as possible [1]. The main component for that kind of factory setting was the concept of Internet of Things where all the different machines in the factory were connected on a network. This connection between everything makes it possible to create a digital twin of a factory which could potentially lead to a more optimized factory through the use of information. With this information the plan was to involve less people in an shifting to a much more heavily automated version of a factory. However, Industry 4.0 was replaced after about 12 years by Industry 5.0 which aims to augment Industry 4.0 by adding the human element back in. One of the goals of Industry 5.0 is the shift back to a more human centric form of automation which empowers workers rather than replacing them [2]. To achieve this the concept of collaborative robots, also known as cobots, is introduced which focuses more on the Human-robot Interaction (HRI) for working together. The idea of working together with robots has emerged in other fields than automation, specifically the health care sector has seen advancements in cobots being used in elderly care and hospitals where they work as personal assistants to the staff [3]. There are two different uses for cobots in the health care, logistic tasks where the robots assist nurses by bringing samples to and from places. The second is social tasks which occurs more in elderly care where the robot fulfil a social need in the elderly. The robots used in this is referred to as service robots.

2.1.1 Service robots

The international federation of robotics define service robots as a robot which is able to perform tasks for humans or machines in a professional or personal environment [4]. To do these tasks the robot requires a number of different features such as mobility, manipulation and external sensors. It is necessary for the robot to be mobile as it moves around the workspace performing tasks for the human. One of the most common tasks for service robot would be a form of item retrieval which requires the robot to be able to manipulate the environment with a gripper. For external sensors there are a few which are an requirement, cameras are essential for the robot to do object detection and manipulation in an unknown environment. The ideal camera for this would be a depth camera for the capability to retrieve real world coordinates. A range scanner for the navigation is also essential, it could technically also be done using a depth camera but it could potentially

lead to issues in dynamic obstacle avoidance. Lastly while not needed being able to use sound for the human interaction can be very beneficial as the most natural method of communication for humans are through speech.

Name	Camera	Microphone	Manipulation	Speaker	ROS
Pepper [5]	2xRGB + 3d depth sensor	True	2x7 DoF arm	True	True
TIAGo [6]	RGB-D	True	7 DoF Arm	True	True
Hobbit [7]	RGB-D	True	5 DoF Arm	True	True
Fetch [8]	Primesense 3d sensor	False	7 DoF Arm	False	True

Table 2.1: Table of four different service robots. The usage of these robot varies with robots like Pepper and TIAGo seeing more use in hospitals, Hobbit being used in elderly care and the fetch robot in warehouses. Features important to a service robot is shown in this table

Pepper

The Pepper robot is a humanoid robot which was developed by SoftBank Robotics. As can be seen in Table 2.1 Pepper has all of the features outlined previously such as microphone and speech capabilities for human robot interaction. For manipulation Pepper has a dual arm setup with sixDegrees of Freedom (DOF) in each arm one in each gripper. The microphone arrays allows humans to use speech to communicate with it as well as sound localization which has been used in hospitality tasks.



Figure 2.1: Softbank Robotics Pepper[9]

TIAGo Robot

The TIAGo robot is made by PAL Robotics and is essentially made with the purpose of being a service robot. The TIAGo is a mobile manipulator well designed for the service robot task. It is equipped with a differential drive base, seven DOF manipulator where six are in the arm and the last from the gripper, RGB-D camera and two microphone arrays [10].



Figure 2.2: The Tiago robot with a omni directional drive base, equipped with a RGB-D camera and a 7 DOF manipulator.

These different tools makes the TIAGo an ideal choice for a service robot, in addition to this the TIAGo has been used in a number of studies where it has been used for various task such as high level user instruction [11] and robotic assistant [12].

Hobbit

Hobbit is a service robot designed for elderly care. It slightly leans more towards an assistive robot but still has built in features such as item retrieval and move commands [7]. The main form of interaction is through its graphical user interface. For studies Hobbit has mostly seen use as a personal assistant in elderly homes where it provides a variety of different features such as safety checks, object transportation, assistance with fitness and the ability for the user to teach the robot about objects of interest [13].



Figure 2.3: The Hobbit Robot. This robot is specifically designed for elder care. [7]

Fetch

Fetch from Fetch Robotics is a service robot which leans more towards working in a warehouse environment due to its bulkier build compared to the previous robots. While it has the manipulation and sensor capabilities it does lack the ability to hear sound. This makes it not possible for the a operator to interact with the robot throuh speech.



Figure 2.4: Fetch Robotics' Fetch Robot with an additional mobile base which carries items picked up by the main robot. This robot is mainly designed for working in warehouses.[8]

These four different robots have been used for a variety of service tasks throughout the years. The Hobbit robot is significantly older than the three other robots and mostly used in scientific research therefore it will not be used. For the three other robots Fetch, Pepper and TIAGo, they are all very similar in the capabilities however Fetch is missing the capability to hear sound. Manipulation-wise this does not matter, but for the interaction between the operator and the robot this is a significant feature. Pepper and TIAGo has seen more use in scientific studies compared to Fetch, specifically in service robot tasks. Between Pepper and TIAGo it is very task dependent, Pepper can in theory handle more complex manipulation task through the use of its dual manipulator setup. The TIAGo is still capable of a majority of manipulation tasks and has a longer reach than Pepper. Both robots could be used for context aware navigation, therefore moving the TIAGo will be considered the chosen robot. The question is then how the robot handles adding context from a human into the problem of navigation. To do this the robot needs a higher level of understanding of its environment in the form of a semantic map.

2.2 Semantic mapping

A semantic map can be split up into three distinct layers, the geometric layer, object layer and scene layer. The geometric layer is represented in the form of an occupancy grid where specific cells are marked as either an obstacle or free space. The geometric map can be created with the mapping algorithms, which can be provided through existing Robot Operating System (ROS) packages such as Navigation2 (NAV2). The object layer contains the different objects found around the mapped environment. To build the object layer an object detection algorithm is needed, combining it with pose estimation and the localization provided by NAV2 allows the system to locate objects in the geometric map. Lastly is the scene layer, this layer uses both the geometric layer and object layer to essentially determine specific scenes in the map. To do this, the objects found in the object layer needs to be clustered together using a clustering algorithm. Once objects have been clustered together the system essentially has a list of poses and object labels. Passing the object labels through a ontology database, which describes relations between things, a label for the location can be inferred and then the geometric placement of the location can be computed using the poses of the objects. The three different layers are visualized in Figure 2.5.

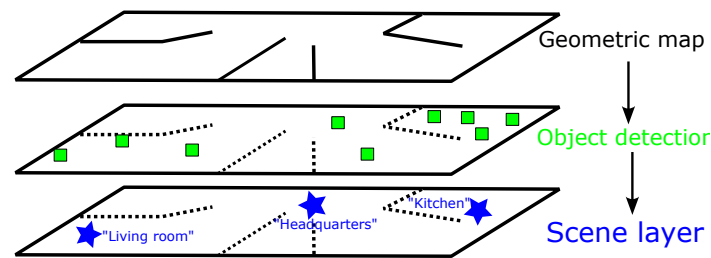


Figure 2.5: Semantic mapping built up on multiple layers, first layer is the Geometric map: here the robot gathers all (x,y) coordinates of the environment like (object, obstacles, wall, doors, etc.). Once the coordinates are found, the second layer will do object detection to find all the objects/obstacles in the rooms. At the last layer, the information gathered is used for scene localization, where the semantic mapping tries to figure out what this place is whether it's a "kitchen", "living room", etc.

To sum up the semantic mapping pipeline, the goal of semantic mapping is first to find all the (x,y) coordinates of objects in the environment. Then, using those coordinates of the objects along the labels to identify what the location is ("Kitchen", "Living room", etc.) and estimate the area of the location. As mentioned, an essential component of semantic mapping is an object detection algorithm which utilises image segmentation to identify and label each pixel[14], [15].

To have a deeper understanding of semantic mapping, Sünderhauf et al. introduced a system which combines semantic place categorisation, mapping and real-time robot operation. By using a combination of a Convolutional Neural Network (CNN) for categorisation and One-vs-All Classifiers for recognising new labels. The One-vs-All Classifiers are used to train one binary classifier per class which each binary classifier tries to answer "is this input an instance of class X or not". For example, a label output is given as "cat"; the classifier says yes if it's a cat; otherwise, "no" [16]. To reduce overlapping of classes, a Bayesian Filter is used to ensure temporal coherence in place classification. Sünderhauf et al. indicated by implanting semantic mapping into their system, where it demonstrated the ability to classify places in environments without needing specific training data and adapt to new categories online. The integration with Bayesian filters provided stable, temporally coherent results, and semantic maps proved useful for robotic tasks like object detection and path planning [14].

Where Kostavelis and Gasteratos et al. used a different approach, they used methods for semantic mapping using metric, topological, and topometric maps as well as vision-based techniques. These approaches are based on their scalability, temporal coherence, and topological map usage. The paper is more about surveying existing methods and categorizing them based on the underlying characteristics. It looks at how semantic maps

can be used for task planning, navigation, and human-robot interaction. They provide a taxonomy of methods, including vision-based systems, laser scanners, SLAM, and topological maps. Their focus is not on a specific robot or approach but rather on the various ways semantic information can be incorporated into robot mapping and navigation [17].

The main goal of semantic mapping is to operate in human-centred complex indoor and outdoor environments, the robot must understand its surroundings, which goes beyond a simple algorithm for avoiding obstacles and building world maps. To handle this complexity, the robot must be capable of extracting semantic information about the environment it is placed in. Typically, initialize localization for navigation ("Where am I?") is not enough for semantic mapping but also ("What is this place like?") is necessary. This gives higher-level information about the environment and decision processing, enabling easier interaction between the robot and humans [14].

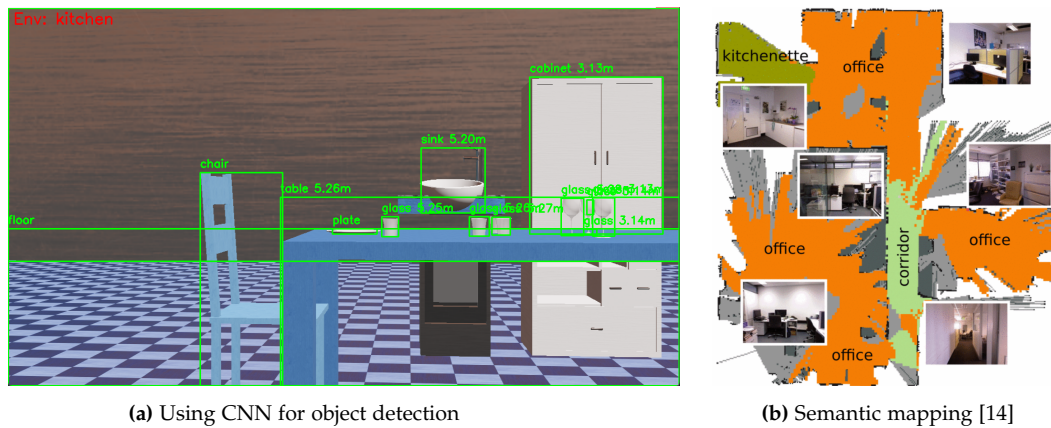


Figure 2.6: The figure shows how the object has been labelled to identify ("What is this place like?") which is then used for labeling the environment.

Figure 2.6 shows the use of a CNN, which classifies each object and the image individually using techniques like one-vs-all and Bayesian filters, gradually building a map based on the resulting place labels. Semantic mapping can be split into three parts. First, "*Understanding the environment*". The main goal of understanding the environment is to know the meaning of different regions when mapping. In this case, the Tiago-base robot has to not just understand the collection of obstacles and free spaces, but also assign meaning to the different regions that have been mapped, like "kitchen," "living room," office, etc. The second part is "*Object Recognition & Classification*". The goal is to identify objects in the environment, such as furniture, doors, and pathways, which would make Tiago-base navigate more effectively. Lastly, "*sensor fusion*". Combining low-level metric maps, such as LiDAR scans, with a high-level semantic understanding of "this is a table" improves navigation efficiency. Once semantic mapping has been completed, a file with ontology

will be made to fill in all the data of the environment in terms that are meaningful and understandable [14], [15], [18].

2.2.1 Pipeline

As described in section 2.2 there are a few different components which form the semantic mapping pipeline, before looking at how the semantic map is used the function of these different components needs to be clarified. Before semantic mapping can be done the robot need knowledge of the environment in the form of a geometric map, this can be done using Simultaneous Localization and Mapping (SLAM) toolbox with the NAV2 package. Once the geometric map is made then additional information can be added to this in the form of semantic mapping.

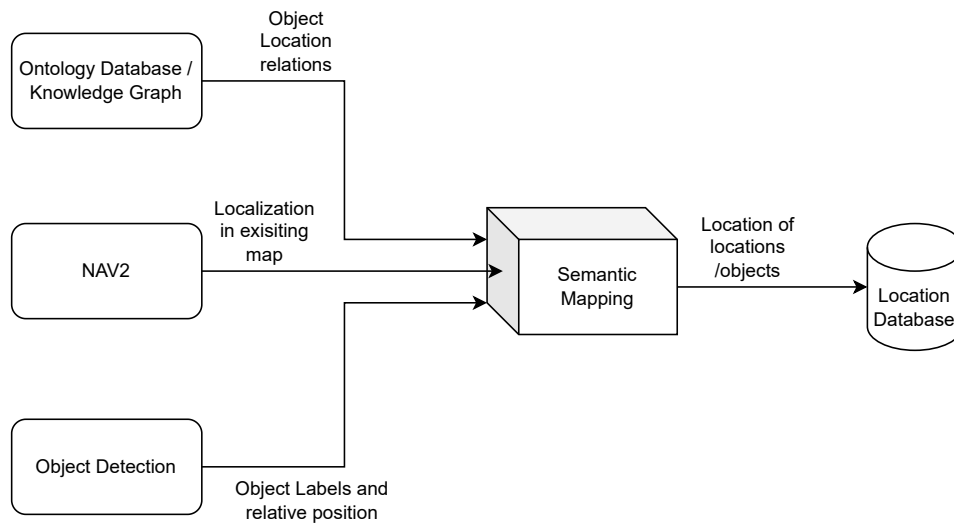


Figure 2.7: Semantic mapping pipeline. The semantic mapping node will take it some localization transform from NAV2 which it will use to estimate the position of detected objects in the environment. Based on the object seen some location is determined, the relations between objects and locations is defined in an ontology database / knowledge graph. Combining this information into location and object positions in the map is then stored in a location database.

Figure 2.7 represents the pipeline for creating the semantic map. The semantic mapping module will combine localization in the map with object detection to find different objects and estimate what location the objects are in. This is combined with information about object relations in the form of an ontology database so the system can estimate where specific locations could be. The output of the semantic mapping is the coordinates of these locations which are estimated. Once the semantic map has been built it is only a question of how the robot makes use of it.

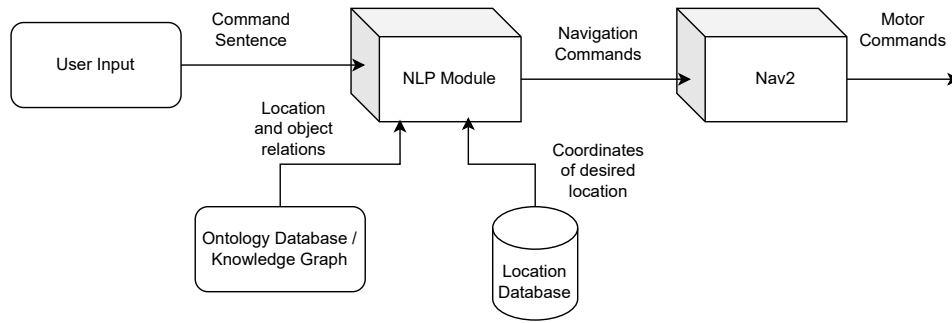


Figure 2.8: The pipeline for handling the different levels of commands. The entry point is in the form of some user input, this could be text or speech. First is the NLP node which have the output of navigation commands. This is done by first extracting keywords such as a task and location/object. Using these keywords it estimates which location to lookup in the database using the ontology database which were also used for the semantic mapping. From the location database it retrieves the map coordinates to go to which is then compiled into a navigation message which is then sent to the NAV2 framework.

Figure 2.8 represents the main loop of the system, for this pipeline it should already have a geometric and semantic map of the environment. The main issue this pipeline tackles is turning the commands from the operator into something the robot understands. As can be seen in Figure 2.8 the pipeline consists of different nodes which fulfil the different capabilities needed. First is the NLP node which is responsible for interpreting the user input and translate it into commands. The NLP node will interface with the database which contains the different coordinates of the locations of interest. The NLP module figures out what data to lookup in the database based on the keyword extraction which is run through the same ontology database.

2.2.2 Image segmentation

Segmentation is an end-to-end process, meaning that a single model handles the entire workflow from raw image input to final output without requiring separate manual stages. During training, the model learns all necessary steps, such as feature extraction, classification, or segmentation. There are three types of image segmentation tasks semantic, instance, and panoptic segmentation, which classify input images based on attributes like colour, contrast, placement, and other characteristics [19].

Semantic segmentation: is a computer vision task used in various fields such as autonomous driving, medical imaging, and robotics, which require a detailed, pixel-level understanding of an image. Its goal is to categorize and label each pixel in the image according to its class and label. A common approach is to use a type of Convolutional Neural Network (CNN) known as a Fully Convolutional Network (FCN). Unlike traditional CNNs that end with fully connected layers and output a single label for the entire image, the FCN relies solely on convolutional layers. This design preserves the spatial structure of the input, enabling the network to generate detailed, pixel-level predictions,

as seen in Figure 2.9, showing how each pixel is semantically segmented [19][20].



Figure 2.9: This image shows how semantic segmentation has categorised each pixel into regions based on the pixel labeling.

Instance segmentation: goes a step further than semantic segmentation. While semantic segmentation labels every pixel in an image with a class, instance segmentation not only does this but also distinguishes between different instances of the same class. This means that if there are multiple objects of the same type (like several cars or people), each one is individually identified and segmented. Instance segmentation provides a more detailed understanding of an image by isolating and segmenting each object instance individually, offering a finer granularity of analysis, as seen in Figure 2.10 how each object has been instance segmented into an individual class [21][22][23].

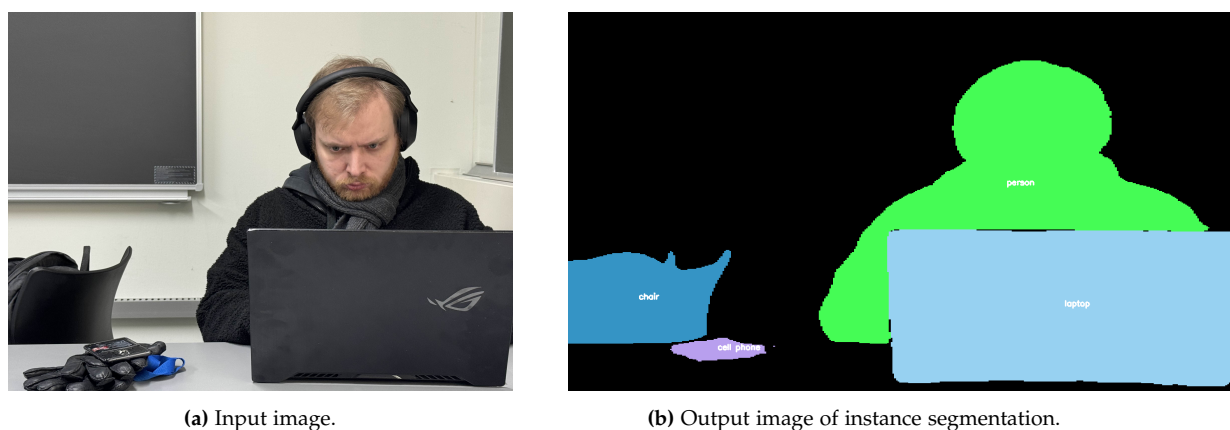


Figure 2.10: This image shows how instance segmentation has categorised each pixel and distinguishes between different instances of the same class.

Panoptic segmentation: Combine semantic and instance segmentation into a single

task. By working with a context-based robot, panoptic segmentation gives both "what is this?" and "which exact object is it?". The robot needs context of semantic to understand the environment, like knowing there is a floor and there is a wall. Also, the robot needs to have instance-level understanding to interact with individual objects, like knowing what an object is, table one, and which one is table two. If only using semantic segmentation, the robot can not tell the difference between tables apart, it would just know it is a table. Also, if only using the instance segmentation the robot can distinguish between cups, but might not understand where the floor or wall is located [24][25][26].



Figure 2.11: This image shows how Panoptic segmentation using both approaches to identify and labeling objects in the image.

2.2.3 State-of-the-art CNN

A typical Convolutional Neural Network is based on a sequence of layers including convolutional, activation, pooling, and fully connected layers, which work together to extract and process spatial features from input data, as seen in Figure 2.12.

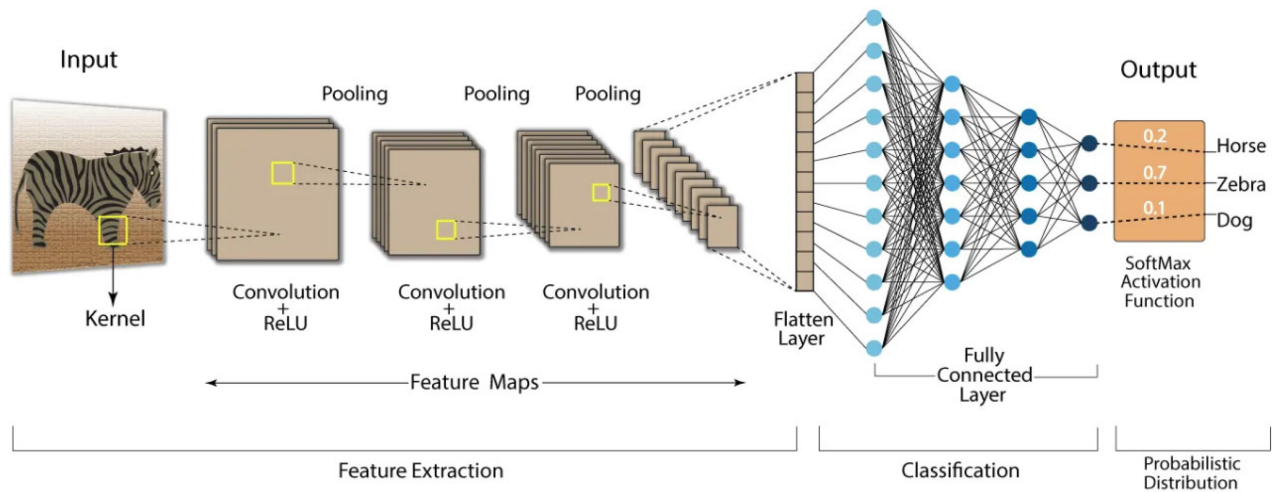


Figure 2.12: A typical model of CNN backbones [27]

Within a backbone network, which is pretrained on a large dataset and optimized for feature extraction, such as AlexNet, VGG, ResNet, DenseNet, and others. They provide the foundational feature extraction capability required to process complex visual data. The selection of the backbone is critical to the performance of the model, as different architectures are designed to address specific challenges in computer vision tasks, such as recognizing objects, classifying images, and segmenting scenes [28].

A table has been set up to show a comparison of state-of-the-art panoptic segmentation methods used in modern research and industry, as seen in Table 2.2. The primary goal of this table is to provide an overview of the different techniques used for panoptic segmentation, showcasing the methods' performance in various aspects such as accuracy, computational efficiency, and ease of use. These methods are evaluated based on key metrics, including Panoptic Quality (PQ), Average Precision for panoptic segmentation (AP_{pan}), mean Intersection over Union (mIoU), the number of parameters (#params), Floating Point Operations (FLOPs), and frames per second (fps).

Method	Year	Backbone	Dataset	PQ (Approx.) (%)	AP_{pan} (%)	mIoU (%)	#params (Million)	FLOPs (Giga)	fps
MaX-DeepLab [29]	2021	MaX-Large	COCO	51.3	57.2	42.4	451	7384	131
Mask2Former [30]	2022	Swin-Large	COCO	57.8	61.7	62.4	216	868	4
Panoptic SegFormer [31]	2022	ResNet-50	COCO	49.6	45.6	47	51	214	7.8
MaskDINO [32]	2022	ResNet-50	COCO	53	46	48	52	285	14

Table 2.2: Comparison of selected panoptic segmentation methods with approximate PQ scores, AP, mIoU, parameters, FLOPs, and fps on COCO.

- **Method:** Lists the name of the panoptic segmentation approaches. The method shows earlier to more recent advances in the computer vision field.
- **Year:** The year of the method's publication. This helps track the progression of

panoptic segmentation techniques over time.

- **Backbone:** The backbone network used for feature extraction. This is crucial because the choice of backbone affects the overall performance and computational complexity of the model.
- **Dataset:** The dataset on which the method was tested. Most methods in this table have been evaluated on the COCO dataset, a standard benchmark for panoptic segmentation, but some are also evaluated on other datasets like Cityscapes.
- **PQ (Approx.):** Panoptic Quality, an approximate measure that combines segmentation accuracy and object detection quality. This metric is essential for assessing the overall performance of panoptic segmentation methods.
- **(AP_{pan}):** Average Precision for panoptic segmentation, focusing specifically on the quality of both instance segmentation and semantic segmentation tasks. It provides a more detailed measure of the model's performance in distinguishing objects and segments.
- **mIoU:** Mean Intersection over Union. This metric evaluates the quality of segmentation at the pixel level, with higher values indicating better segmentation performance.
- **#params:** The number of parameters in the model, which is a measure of model complexity and its potential impact on inference time and computational resources.
- **FLOPs:** Floating Point Operations, which indicate the computational cost of the model. This is important for evaluating the efficiency of the method in terms of required computation.
- **fps:** Frames per second, a measure of inference speed. This is critical for real-time applications where fast processing is essential.

Based on the table and various aspects such as Panoptic Quality (PQ), Average Precision for panoptic segmentation (AP_{pan}), mean Intersection over Union (mIoU), the number of parameters (#params), Floating Point Operations (FLOPs), and frames per second (fps), an appropriate method can be selected according to the project's needs. When picking a method, the focus will be on three metrics Panoptic Quality (PQ), Average Precision for panoptic segmentation (AP_{pan}), and mean Intersection over Union (mIoU).

2.3 Understanding humans

Once the semantic map is built and ready to be used the system needs to consider how to translate human commands into robot actions. The first step is to actually get an objective

for the system from a human user. There are multiple ways of doing this, it could be done through a text prompt or graphical user interface but humans' preferred method of communication is through speech. This is something that has been explored plenty through out the years with multiple methods of doing this. This chapter will not focus on converting speech into text but what methods is used to understanding the speech which have been processed into natural language in the form of text, this is known as NLP. Studies on NLP often uses a variation of the Recurrent Neural Network (RNN), the main reason for the use of RNNs is the how it excels in handling sequential data [33]. When humans look at a sentence their understanding of the meaning depends the current word being read and the words prior. In neural network fashion one could say that there is an internal state which is updated with each data entry being processed. This is how an RNN works, it receives a single data input at a time and updates its internal state. There are multiple different variations which solves issues such as exploding/vanishing gradient which affects the baseline RNN architecture.

2.3.1 Keyword Extraction

When looking at a sentence there often are "padding" words which helps form the simplest command into a coherent sentence for humans. Consider the following command "Please bring me a box". Looking at the sentence it is easy to split it into an action and a target for the action. In this case "bring" is the action of retrieval and box is the target for the action. The question is then how does a computer understand which parts of the sentence is what. To do this there needs to be some form of keyword extraction. The most common method for keyword extraction is the use of neural networks, specifically variations of RNN has been widely used. More recent state of the art however moves away from the use of RNN in favour of transformers.

Long Short-Term Memory (LSTM)

One of the RNNs which often used is a LSTM as seen in the following studies [34] [35]. The LSTM is a RNN. There are few different variations of this model but one of the more commonly used is the bidirectional LSTM model. The main strength of the LSTM model is the ability to retain information while processing large inputs, however there are some shortcomings. The main issue with the standard LSTM is the way sentences sometimes provide additional information at the end. Consider the following sentence: "Please bring me a cold glass of water and not ice water.". The standard model in this case first recognizes it needs to bring a cold glass of water which could potentially mean a glass of ice water as it is cold. With the bidirectional LSTM the sentence is processed from both sides, and in this case the robot would have the context of it needs to bring cold water which is not ice water.

Gated Recurrent Unit (GRU)

State of the art shows that the GRU model has seen uses in the natural language processing of voice commands for medical robots [36]. GRU is similar to the LSTM model, the main difference is in the way the hidden state is updated. LSTM uses updates its memory using the forget and input gates while GRU achieves this with only one and update gate. This gives a slight advantage in real time performance and training time of the model however, the model performs slightly worse than the bidirectional LSTM model. With the introduction of transformers both GRU and LSTM was used less

Bidirectional Encoder Representation from Transformers (BERT)

BERT moves away from RNN models over to transformers which makes use of the attention concept discussed in the Attention is all you need paper[37]. An advantage of transformers is in the time to train, RNN can only handle data sequentially while transformers is able to process the input data in parallel. In addition to being more easily trained it also holds an advantage over the RNN models mentioned when dealing with very large inputs.

Summary

While Transformer models such as BERT is the current state of the art for NLP modules given the short commands defined in 2.3 all of the different methods could potentially be used in the use case. However since there w

2.3.2 Ontology

When the keyword extraction is done it should leave keywords in the form of an action and a target, the target however is not always clear cut for the robot. Consider the input "I am thirsty can you bring me something". Humans would easily recognize that the person wants something to drink, robots would not understand what the target here is specifically. The system needs to have some sort of knowledge of how different words relate to each other. There are multiple ways of imbuing the robot with knowledge, a standard of doing this is through the use of an ontology database such as Web Ontology Language (OWL) [38]. This is a method which has seen use in previous studies of semantic mapping [39]. There are also other methods for knowledge learning, with advancements in Large Language Model (LLM) there have been studies where the knowledge section of the robot is using LLMs for translating commands into robot commands [40, 41, 42]. While an knowledge graph is good and fast for development of specific use cases it struggles with scalability as adding more objects and relations between objects, locations and needs is exponentially more work and will have issues in environments which are dynamic.

2.3.3 Navigation

Since the robot will need to move around the environment there needs to be a navigation system which can interface with the NLP component of the system. Considering that the TIAGo is built using ROS2 framework and already have implementations in NAV2 available it makes sense to use the NAV2 for the navigation system. NAV2 contains all the necessary functionality needed to run a navigation stack. This involves components such as localization, path planning, path execution and behaviour planning in the form of behaviour trees. The challenge is in converting the processed input from the user to commands which the framework understands. NAV2 mainly works with poses in a known environment which have been mapped out using SLAM algorithms such as Cartographer and Gmapping. Using the semantic map the system has specific areas of the geometric map which can be used to compute navigation poses. The question is then what kind of commands can the system expect?

2.4 Problem / Use case

Before looking at the design of the specific components in the system it is important to describe the problem. In this case the system will be a robotic assistant which is able to help the user. The first task that will be considered is a retrieval task. There are multiple levels to retrieval task, consider the following tasks.

Level 1:	<i>Go to the kitchen</i>
Level 2:	<i>Go to the kitchen and bring a glass</i>
Level 3:	<i>Bring me a glass</i>
Level 4:	<i>I am thirsty</i>
Level 5:	<i>Please assist me</i>

Table 2.3: Overview of the different levels of commands. Each level comes with a new level of requirements. Level 1 covers the basic command to navigation. Level 2 introduces commands with sequentiality, the robot first needs to drive to the kitchen, find a glass and bring it back. Level 3 brings an increased need for semantic capability to the system, while the command semantically means the same as the level 2 command the robot needs to be able to understand a glass can be found in the kitchen.

For humans the most natural way to communicate is through the use of their voice. The problem is how can the robot hear a command, translate it to robot commands and then execute it. There are several levels of commands as can be seen in Table 2.3.

Level 1 is a mid level command, it simply saying to the robot it should drive to a specific location. Of course there needs to be some understanding of where this location is in the natural way of describing the environment. In the case of level 1 the robot would simply have to convert the location "kitchen" into some 2D coordinates for the robot.

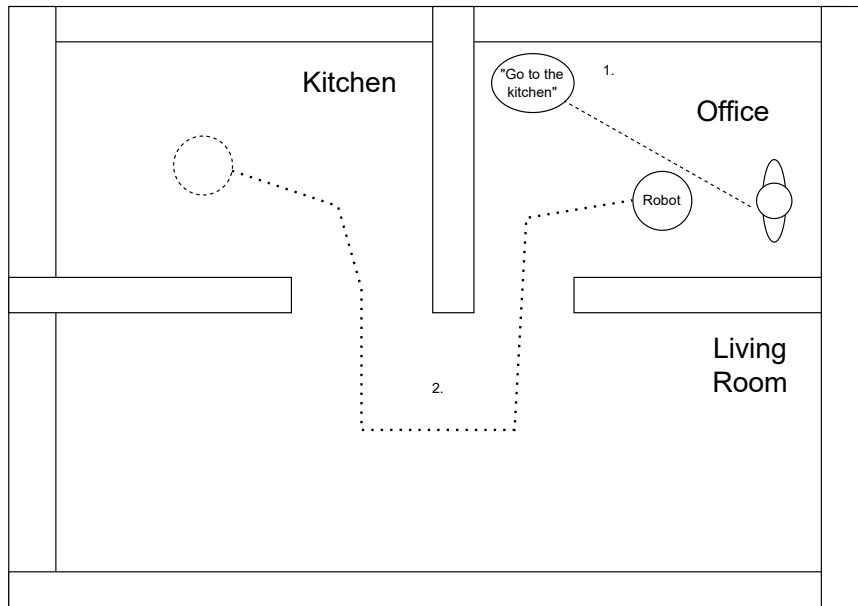


Figure 2.13: Example of a level 1 case, 1. The robot is given a command from a user, in this case the user is in the office and the command is to go to the kitchen. 2. Navigate to a pose in the kitchen.

Li et al. [40] explored commands similar to what described here. While level 1 describes a location it could also be something simple as an object.

Level 2 The second level bring additional information to the level 1 prompt, in this case the robot needs to understand it first needs to drive to the kitchen, then find a glass and bring that glass back to the user. The main challenge here is how the robot actually finds the item. A study [43] looked into a similar problem in the form of the prompt "pick up the black cup". Li et al. also solved similar issues by taking the high level command and splitting into smaller tasks [40]. Looking at the level 2 tasks it could be split up into the following smaller tasks:

1. Navigate to kitchen
2. Navigate to glass
3. Grasp glass
4. Navigate to start position
5. Place glass

Level 3 is closer to what could be considered normal speech pattern, level 1 and 2 leans a little more to specific instructions while level 3 essentially is a higher level instruction

which might fulfil the same purpose as the level 2 command. In the case of level 3 the robot essentially needs to be able to understand where it can find a glass. There are several methods for making this connecting between glass and the location it could be located in.

Level 4 removes the specification of what the operator actually wants. When considering the input "I am thirsty" humans would of course think of a glass of water to quench the thirst. To be able to solve this problem the robot would need to be able to understand who the target is, and what item they actually want. Li et al. [44] used a similar command for a pick and place operation where the robot based on this command predicted which item on the table in front of it to pick up. The method for extracting information from natural language was through the use of rule matching and Conditional Random Fields. Combining the natural language processing with image recognition the robot was able to predict which object to grasp based on the command. The question is then can the same method be used on a service robot.

Level 5 is the level which completely removes any given context from the user. The robot would need to deduct the context of the situation based on its own sensors. To achieve this it would require a very complex system which is able to make use of all the different sensors on the robot.

3 Problem Formulation

After exploring the problem the requirements for a system which uses semantic mapping for context aware navigation is much clearer. Semantic mapping requires the robot to be able to build the three different layers described in section 2.2. This involves the use of laser scans, object labelling and pose estimation in the environment. Using the information found by these components the system needs to infer the specific scenes found in the geometric map. To apply the context to the navigation the system requires a form of natural language processing which is able to understand high level commands from the operator and convert it into commands for the navigation system. Finally these different components of semantic mapping has been presented in a pipeline which describes the data shared between them. This leaves the final problem formulation.

3.1 Final Problem State

"How can a context aware navigation pipeline be used in a personal service robot for object retrieval tasks through the use of natural language processing and semantic mapping?"

3.2 Objectives

1. **Complete Level 4:** As described in section 2.4 there are several levels of commands which require different things of the robot. To make sure that the objective is feasible within the scope of the project the robot should be able to complete up to level _
2. **Interpret Commands:** The ability to take high-level instructions from the operator and turn it into input for the rest of the system is essential for this project. Specifically the system should be able to identify actions and targets of those actions. In addition to this being able identify multiple actions and targets and handle them sequentially will also be very helpful.
3. **Create a semantic map:** To be able to interpret high-level commands from the operator the system needs a higher understanding of its surroundings. This will be achieved through the use of cameras to detect the environment. The goal is to build the three layers described in section 2.2.
4. **Identify object in the environment:** To do semantic mapping the system will need to be able to identify objects within the environment. This is done through the use of an object detection neural network. The objects and location of detection is essential to be able to semantic mapping.

5. **Ready to grasp an object:** The final goal of project is for the robot to be able to retrieve the desired object from the high-level command. Being able to place the robot at a pose where it is able to grasp the target sets up a clear point where a grasping pipeline can be applied.

3.3 Delimitations

Since there is a finite amount of time for this project there are several delimitations with needs to be outlined.

1. **Simulation:** The goal of making a pipeline which is able to perform semantic mapping will only be implemented in simulation. There are two reasons for this choice, focusing on simulation avoids spending time making the different components of the pipeline work on the robot itself giving. Avoiding spending time on making the components works allows for more time to make the general pipeline work. Secondly having a simulation environment allows for more varied type of environments. Being able
2. **Command complexity:** the complexity level of the commands the robot will be tasked with, the commands described in section 2.4 are simple in the form of being comprised of a single location and target object. These commands could have more added complexity to cover, an example could be "go to the kitchen and bring the glass which stands on a red box". This command adds even more context to the command and it makes sense as a future work goal.
3. **Navigation:** For the navigation the environment which the robot will be moving around in will remain simple so no moving obstacles. There will be no specific goal for the navigation time but mostly whether the robot is able to understand where to go and reach that specific location.

4 Design

This chapter will cover specific design choices for the system, this includes choice of algorithms and methods for the different system components. It will also cover the construction of the simulation environment.

4.1 Simulation Environment

The simulation is made to simulate a real environment, the aim is to have a number of different type of rooms. There are three types of rooms, kitchen, office and bedroom. A room is designated based on the objects in the room as the semantic mapping would understand it. The kitchen is a room where objects such as a fridge, kitchen counter with an oven and stove and some dining furniture. In addition to this several objects which can be picked up is placed in the environment such as cans. The kitchen can be seen in the Figure 4.1



Figure 4.1: The kitchen room environment,

An office space would contain a workspace with objects such as desks, computers, cups and some general office space objects.

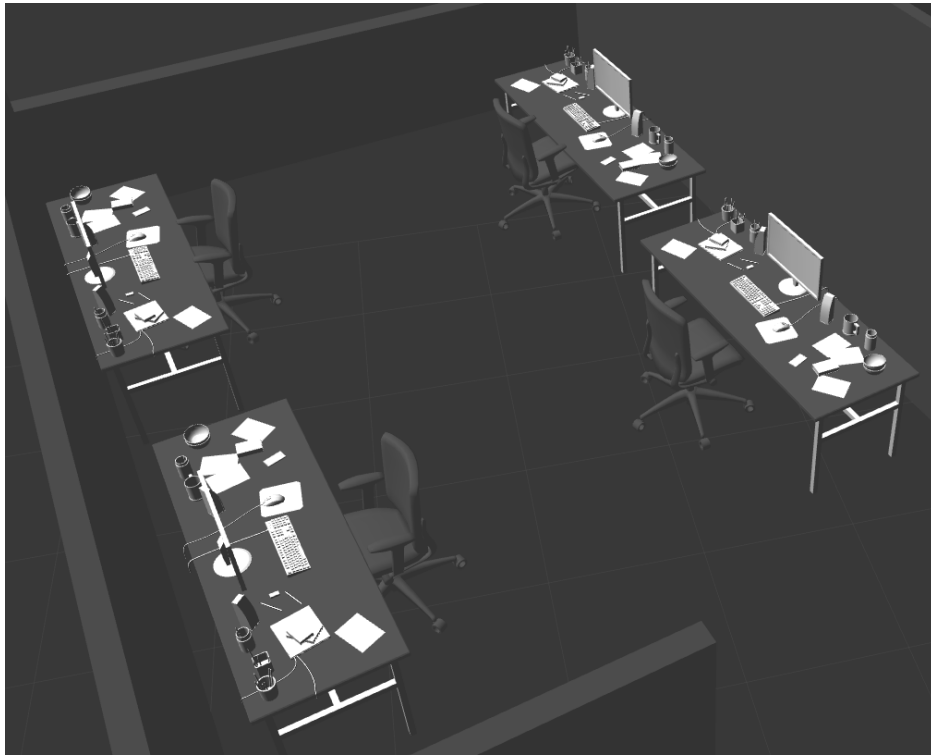


Figure 4.2: The office environment.

Lastly is the bedroom, here usual objects are objects such as a bed, night table with a lamp etc.

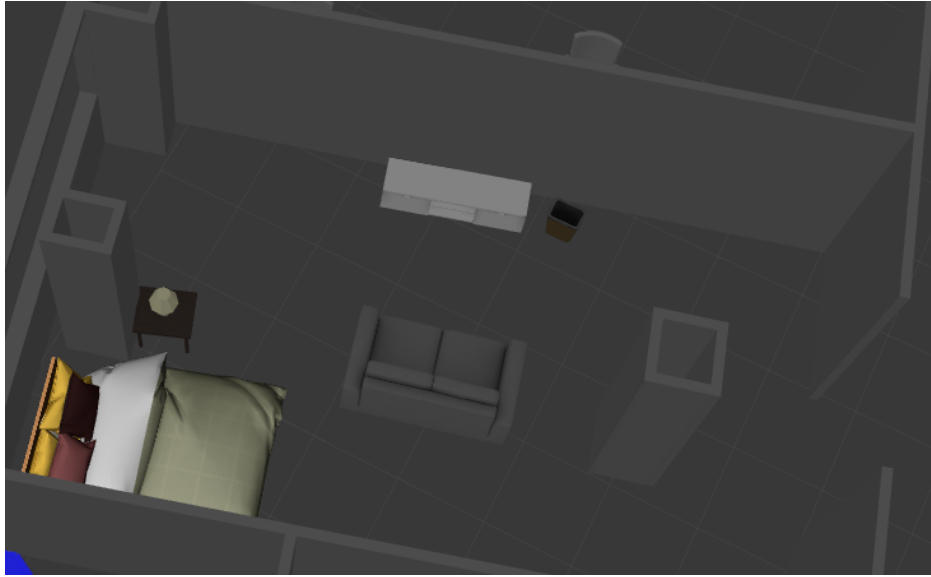


Figure 4.3: The bedroom environment

In total there are the three types of rooms described and five rooms in total. In these different rooms there will be the objects which can not be manipulated by the and small objects spread through out the room.

4.2 System overview

This section will discuss the design of each component in the pipeline described in subsection 2.2.1. Choice of algorithm and the expected input and output of each component will be the main points.

4.2.1 Ontology

For the ontology OWL is being used for the specific problem that is being solved. For that purpose an ontology graph is built which aims to show relations between objects and locations but also relations between objects and needs. First step is to define the different classes, first is location of course which describe different locations in the map. Next class is objects which describe different objects in the environment. Lastly is needs, this describes different wants and needs which might infer needs for specific objects, an example of this is the need "thirsty" which infers something drinkable.

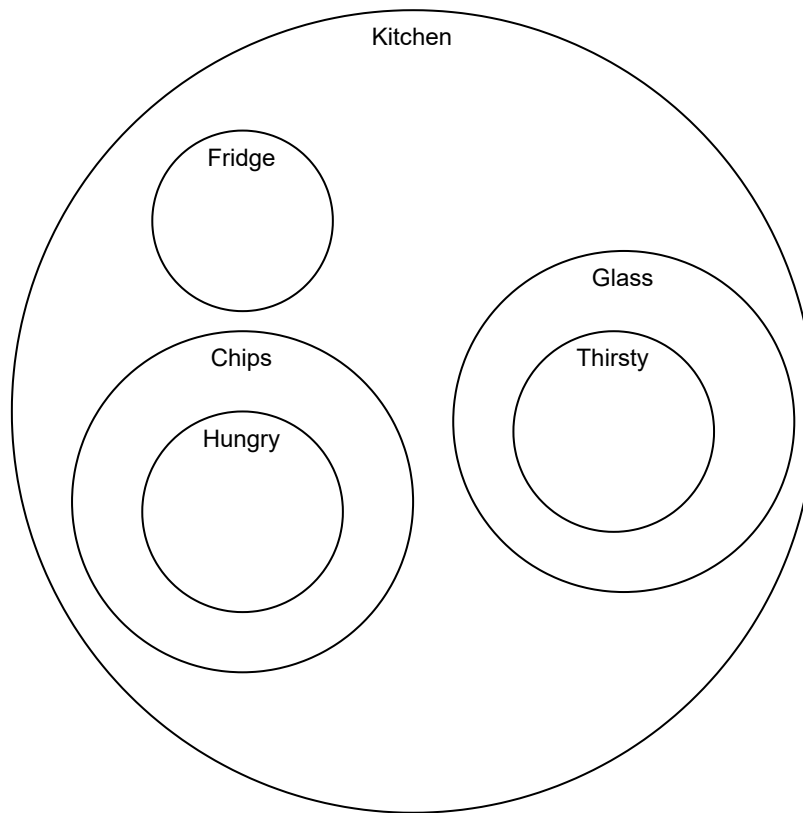


Figure 4.4: Example of an ontology graph. There are three main categories, location, object and needs. As the location will in most cases be the main output of the ontology graph it is made as the root of the graph. From the location specific objects can be inferred, this goes both ways as the location can also be inferred from objects. The class object has a subset class in the form of "needs" where specific objects can fulfil specific needs. This means from a level 4 command such as "i am thirsty" can directly be inferred to specific objects which then can be inferred to a specific location.

An example relation can be seen in Figure 4.4 where a simplified relationship for the location kitchen is described. In this case three objects are defined it be located in the kitchen a fridge, a glass and chips. These objects which can be found in the kitchen has a set of needs attached to them, in this case a glass is something which relates to being thirsty while chips relate to being hungry. Using these relations the location kitchen can be inferred from either an object or a need. As described in subsection 2.3.2 there could also be used something like a LLM. While LLM have seen more use in the recent state of the art this is not implemented in this system for two reasons. First is the computational requirements, the robot is already running a number of different algorithms such as a navigation stack and object detection model. Adding a LLM on top of the necessary components can

potentially strain the performance of the robot. This could be circumvented interfacing with a server which runs the LLM, but this gives the robot a requirement of a constant internet connection with the server which can limit the amount of environments where it could be deployed. LLM is much better for scalability as it is able to apply reason to a much larger variety of inputs.

4.2.2 Natural Language Processing Node

This node is responsible for converting the command from the operator to robot commands. As explored in subsection 2.3.1 there are three commonly used architectures for processing the command. The most recent state of the art has primarily been using Transformers, while a LSTM or GRU model certainly could be used for the commands described in section 2.4 this node will make use of a transformer model. Specifically it will implement the python package Spacy which provides the transformer model roBERTa. The Spacy framework does not naturally provide keywords for location and action to take, therefore this needs to be handled within the node. To do this the node will make use of the tags provided from the Spacy framework, these could as an example be nouns and verbs.

4.2.3 Object Detection Node

This node is used to detect objects by implementing a CNN model, which identifies the objects and labels them. It is not enough to just identify the objects; the accuracy of the identified objects must also be labelled correctly. To address this, in Section 2.2.3, multiple CNN models were shown in Table 2.2. The idea is to use a model with high accuracy and correct labelling, since learning the context of an environment can be taxing. To choose a model, three things must be evaluated: Panoptic Quality (PQ), Average Precision for Panoptic Segmentation (AP_{pan}), and mean intersection over union (IoU).

Upon closer inspection of Table 2.2 minimum and maximum of the (PQ), (AP_{pan}) and (mIoU):

- (PQ) = (49.6%) - (57.8%)
- (AP_{pan}) = (45.6%) - (61.7%)
- (mIoU) = (42.4%) - (62.4%)

Where Mask2Former model seems the most prominent with the highest values. Another key aspect is the position estimation to further the object detection node. By looking into the 3D point estimation using the intrinsic parameters and Depth. The intrinsic parameters defines how a 3D point in the real world is projected onto a 2D image plane. The intrinsic camera parameters describe the internal geometry of the camera, such as [45], [46]:

- Focal length (f_x , f_y)

- Principal point (c_x, c_y)
- Skew coefficient (s)
- Distortion coefficients

Given a pixel coordinate (u, v) , a depth value Z , and camera intrinsics f_x, f_y, c_x, c_y , this can compute into the 3D point (X, Y, Z) in the camera coordinate frame as seen in Figure 4.5

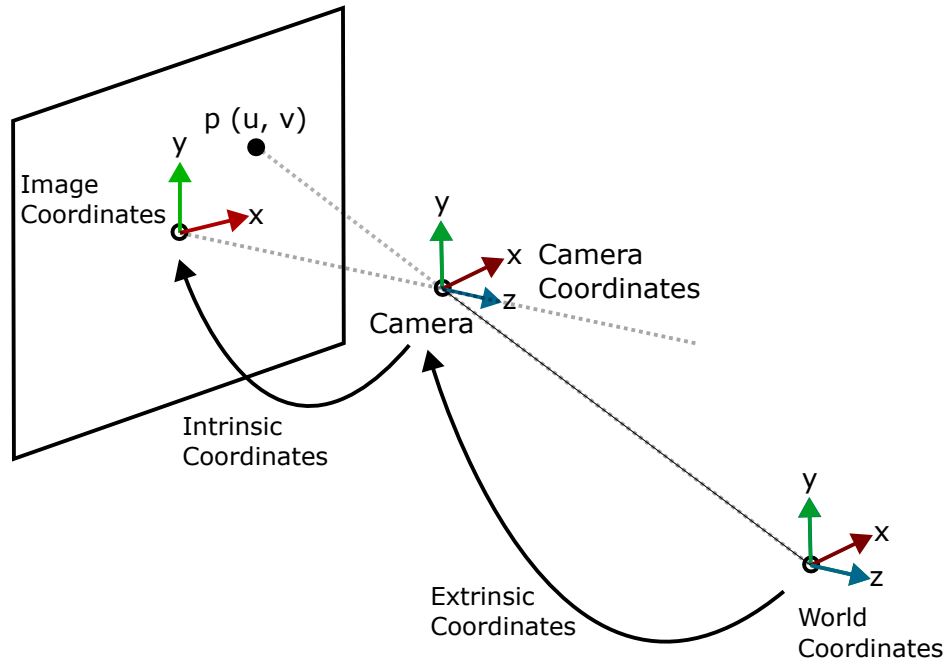


Figure 4.5: This image shows how a point in the world gets projected to the image plane of a camera when doing intrinsic camera parameters [45].

Figure 4.5 shows three coordinate systems and the transformations between them. First, world coordinates, where the global reference frame shows how 3D objects are defined, and a point represents a physical location in the real world. Second, the camera coordinate system, which is centered at the camera's optical center (the pinhole), where the axes (x, y, z) move with the camera. Third, the image or pixel coordinates, which represent the 2D plane where the image is captured. The coordinates are denoted as $p(u, v)$, where u is the horizontal pixel coordinate and v is the vertical pixel coordinate [47], [48].

4.2.4 Semantic mapping node

The purpose of this node is of course to build the semantic map of the simulation environment. To do this it needs to be able to build the different layers described in section 2.2 by performing the following tasks.

1. Record all inputs from object detection node
2. Locate the objects in the global map frame
3. Cluster the objects into specific locations.
4. Compute the areas covered by the clusters
5. Infer the location based on object labels.

The node starts off with the input from the object detection where as described in subsection 4.2.3 it receives a list of object labels and their local poses in relation to the camera. The node is responsible for transforming the local poses so they relate to the global map frame. Once all the objects have been transformed into the global map frame the node clusters the objects, for this a clustering algorithm needs to be chosen. One of the most commonly used methods is to simply base cluster pairing on the distance between objects, this could be achieved using a density based clustering algorithm such as Density Based Spatial Clustering of Applications with Noise (DBSCAN). As for computing the areas there are a two methods which could be used, for a more straight forward area finding the Minimum Bounding Rectangle (MBR) otherwise something like a convex hull for the location could be used.

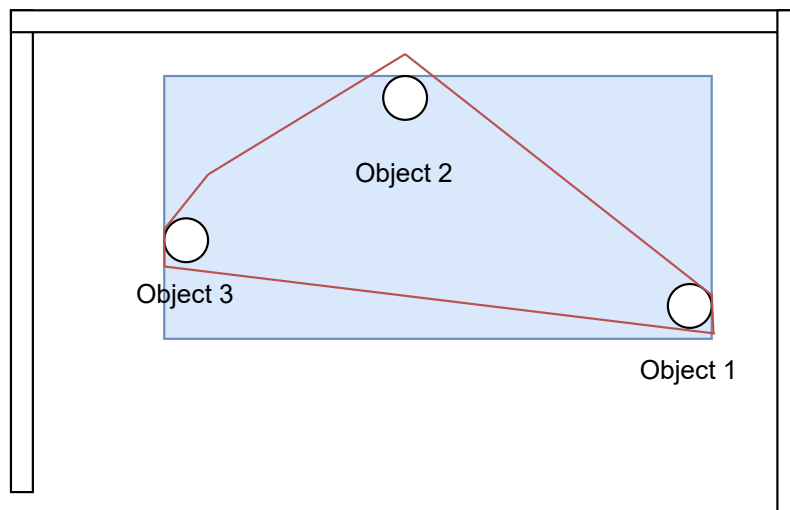


Figure 4.6: Comparison between convex hull and MBR, the convex hull gets a much more condensed area compared to the MBR. The MBR is simpler to describe where you would only need width, height and center coordinates while the convex hull would require storing every single coordinate making up the hull.

The first and the simplest would be computing the MBR, this provides the rectangle which has the smallest area and contains all the different poses of the object. Another

option could be computing the convex hull for all the poses, this gives a convex shape which contains all the different poses. This node will make use of the MBR over the convex hull, the main reason is that the convex hull produces a more complex shape than what is needed for the use case as can be seen on Figure 4.6. Since all the different locations are placed in mostly rectangular rooms the MBR should not be able to overlap into different rooms when computed. It also makes the process of saving the semantic map more streamlined since we do not need to store a complex shape, instead the location can simply be stored by saving the width, height and the center coordinates of the rectangle.

4.3 ROS Node Graph

The goal of this section is to give an understanding of the overall system, how each node talks to each other and what information is retrieved from where and the general output of each node. There will be two different overviews, one for the semantic mapping pipeline and for using the semantic map. First we will look at the semantic mapping overview.

4.3.1 Semantic mapping overview

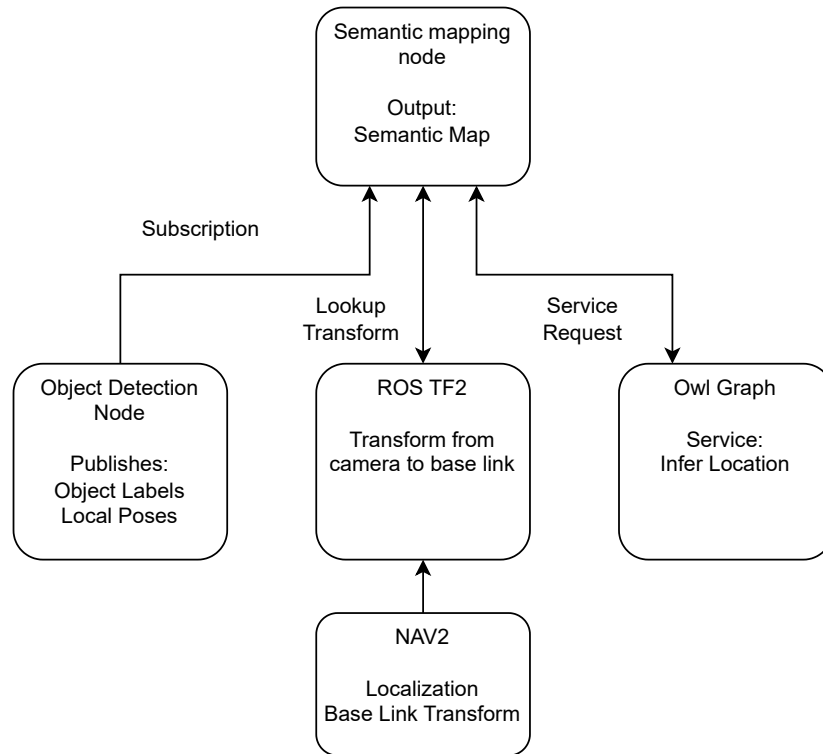


Figure 4.7: Overview of the semantic mapping node graph. The root here will be the semantic mapping node since this is where all the data from the different nodes are assembled into the semantic map. The semantic mapping node receives a list of object poses and local coordinates in relation to the frame from the camera. These local coordinates are transformed to the global map coordinates which is then stored internally in the node. When the operator thinks enough objects have been recorded the compute location service can then be called uses the ontology to determine a location and the object poses to estimate geometric area.

The semantic mapping node the central node in the semantic mapping node. As described in subsection 4.2.4 the node is subscribed to the object detection node which provides a message with the list currently visible objects and their local pose. The semantic mapping node then processes the message from the object detection node, all of the detected poses will be stored locally until finished. The semantic mapping node has an exposed service which can be called to compute the location area. When called it will go through the previously mentioned algorithms to first cluster the objects into locations, compute the MBR of the location and make a service call to the OWL graph to get the location label for each cluster. When these steps are complete the semantic map is saved and can then be loaded using the semantic map server.

4.3.2 Context aware navigation overview

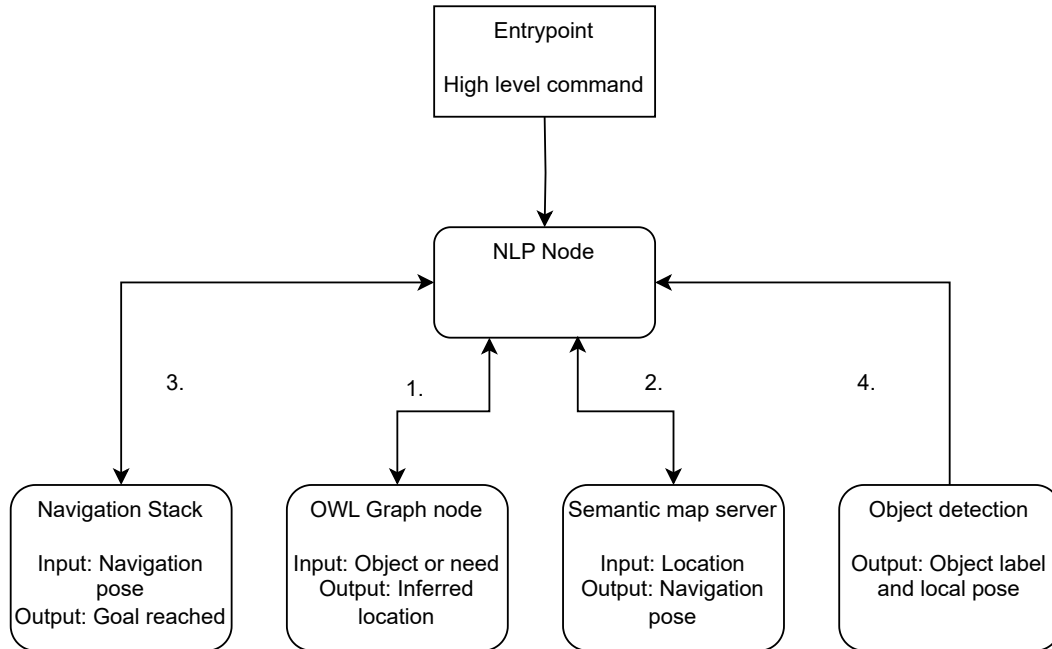


Figure 4.8: Overview of the main loop for context aware navigation. The entry point for the operator in this pipeline is the NLP node. The NLP node is connected to the OWL graph node, the navigation stack, the semantic map server and object detection. The OWL graph is used to infer target locations from the commands if not directly given in the command. Using the target location the NLP node makes a request to the semantic map server for the coordinates of the target locations. These coordinates are then sent to the navigation stack which then navigates to the target location.

For the context aware navigation the root will be the NLP node which subscribes to the a high level command topic which can be used for the input as seen in Figure 4.8. When an input is published the NLP node processes the input into a action and a target, 1. step is, depending on the level of command, the NLP node makes a service request to the OWL Graph node to infer a location based on the a given target from the input. Once the target has been inferred the 2. step is the NLP node makes a request to the semantic map server to get the coordinates for a specific location. Once the NLP node has received a navigation pose from the semantic map server the 3. step is sending a go to pose request to the navigation stack which will then handle the navigation to the target location. The 4. and last step is when the NLP node has confirmation that it is at the location it will use the object detection to find the target object and bring the target back.

5 Implementation

This chapter will cover the specific implementation of the algorithms described in chapter 4.

5.1 Semantic mapping

This section explains the implementation of the semantic mapping algorithm. There are several steps that the algorithm goes through to generate the output, which is a JSON file containing information

such as "pose", "object label", "object position", and more.

The node starts by initializing the semantic mapping process. It subscribes to several topics, including `object_local_pose` and `cmd_vel_out`. These inputs are used to determine whether the robot is moving and to transform object poses into the global map frame. The `compute_location` is implemented as a service within the node and is not an input topic; instead, it serves as a callback trigger that finalizes the mapping process when called.

If the robot is not moving, the node looks through the current object's local pose message and transforms the object poses to the map frame. If the robot is moving, it skips this step. Next, the node checks if the detected object is near an existing one with the same label. If not, it adds the object to `object_poses` and `object_labels`, and the compute location service is only activated by the mapping, which triggers the `compute_location` service. If the object is already near one with the same label, it skips this imaginary step.

Once the object poses have been collected the compute location service can be called where the object poses are clustered, and each cluster of object labels are passed to the `owl_graph` service, which assigns a location label. Based on the location label and the poses of the detected objects, bounding boxes are computed, including their size and centre point. Then, markers are published, and the bounding boxes are sent to the `/location_areas` and `/location_area` topics for visualization. The semantic map is then saved as a JSON file. The node continues running in a loop until shutdown. To better understand the full code, a flowchart was created based on `semantic_mapping_node.py`, which can be seen in figure 5.1.

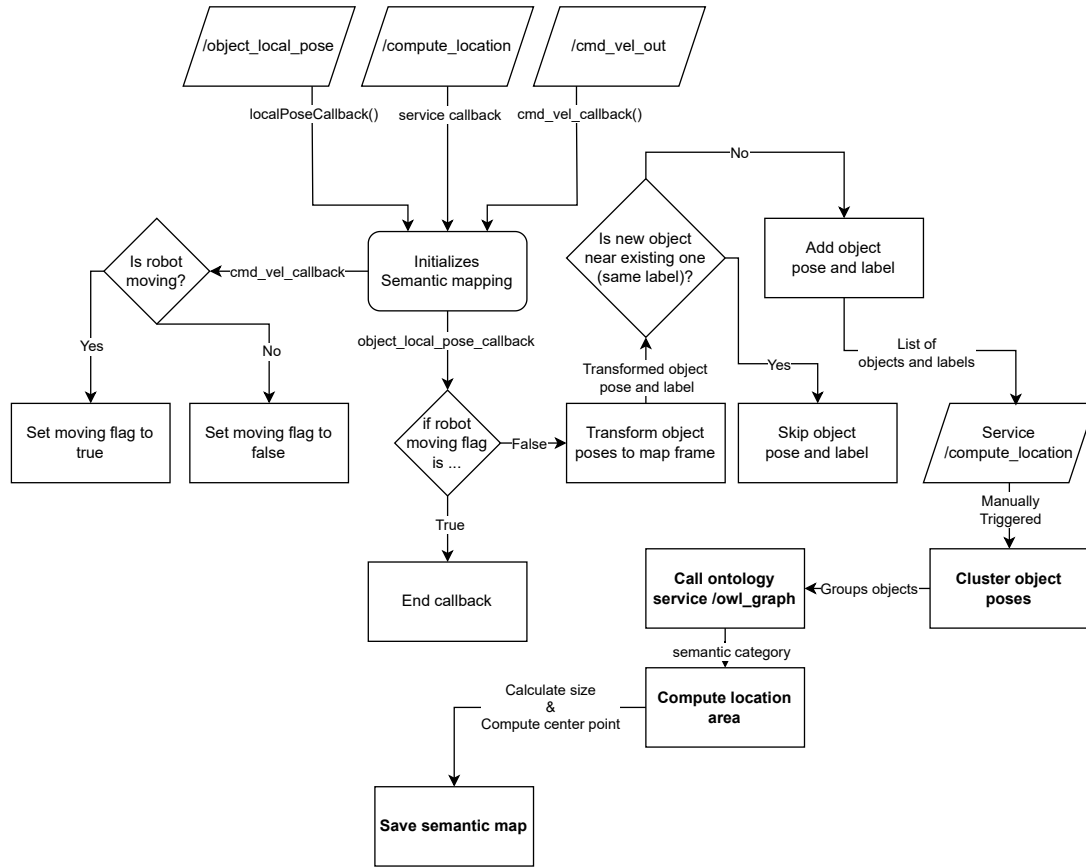


Figure 5.1: The semantic mapping pipeline. There are three separate callback, cmd_vel callback, localPoseCallback and computeLocationCallback. The cmd_vel callback keeps track of whether the robot is moving and sets a flag based on the output. The localPoseCallback is responsible for keeping track of the recorded object poses and labels, in this process it adds objects while the robot is not moving. When adding new objects it checks whether the new object is near an existing recorded object it skips it otherwise it adds it to object poses and labels. Lastly is the compute location callback, when this is called it takes the list of object poses and labels and process them into a semantic map.

5.1.1 Object Detection output processing

As described in subsection 4.2.4, the first step is to process the message received from the object detection node. Where the implementation of object detection in the algorithm is to get some input data from the camera intrinsic parameters node, the input data fetched from the camera is the following includes the RGB image frame, depth image, and camera intrinsics (fx, fy, cx, cy) . The depth image is used to determine the distance to an object, which is denoted as the Z-axis in the camera coordinate frame. The RGB images pass through the Mask2Former model with *"Swin-large-ade-panoptic"*, which extracts the information and outputs the objects and labels. This is used for environment classification

and to find the pose of the object. Camera intrinsics (f_x, f_y, c_x, c_y) are used for this purpose. By looking into the 3D point estimation using the intrinsic parameters and Depth. Given a pixel coordinate (u, v) , a depth value Z , and camera intrinsics f_x, f_y, c_x, c_y , this can compute into the 3D point (X, Y, Z) in the camera coordinate frame using the following transformation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Z \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{(u-c_x) \cdot Z}{f_x} \\ \frac{(v-c_y) \cdot Z}{f_y} \\ Z \end{bmatrix}$$

Where:

- (u, v) is the 2D pixel location in the image
- Z is the depth value at that pixel
- f_x, f_y are focal lengths in pixels
- c_x, c_y are the optical center coordinates

To better understand what happens when running the panoptic segmentation node, an algorithm table which shows the pseudocode of the algorithm [1] can be seen, and a flow chart in the appendix show how everything works together A.1.

Algorithm 1 panoptic segmentation node

Require: RGB image frame, depth image, camera intrinsics (f_x, f_y, c_x, c_y)

Ensure: Updated segmentation image and object poses

```
1: (objects, labels, image)  $\leftarrow$  DETECTOBJECTSFROMCAMERA(frame, depth_map)
2: environment  $\leftarrow$  CLASSIFYENVIRONMENT(labels)
3: fused_objects  $\leftarrow$  FUSEDetections(objects)
4: for all obj in fused_objects do
5:   ( $x_{min}, y_{min}, x_{max}, y_{max}$ )  $\leftarrow$  obj.bbox
6:   ( $u, v$ )  $\leftarrow$   $\frac{(x_{min}+x_{max})}{2}, \frac{(y_{min}+y_{max})}{2}$ 
7:    $d \leftarrow$  obj.distance
8:   if  $d \neq \text{None}$  then
9:      $X \leftarrow (u - c_x) \cdot d / f_x$ 
10:     $Y \leftarrow (v - c_y) \cdot d / f_y$ 
11:     $Z \leftarrow d$ 
12:    obj.position  $\leftarrow$  ( $X, Y, Z$ )
13:   end if
14: end for
15: ANNOTATEIMAGE(image, environment)
16: initialize pose_msg
17: for all obj in fused_objects do
18:   if obj.position is valid then
19:     APPENDPOSETOMESSAGE(pose_msg, obj.label, obj.position)
20:   end if
21: end for
22: PUBLISH(pose_msg)
23: return resized annotated image
```

The algorithm table [1] panoptic segmentation node goes through the following steps.

1. Use the RGB frame and depth map to detect objects and get their labels and a segmentation image.
2. Classify the environment based on the detected object labels.
3. Annotate the segmentation image based on the classified environment.
4. Initialize a new message to hold object poses.
5. For each fused object:
 - (a) If the object has a valid position, append its label and position to the pose message.

6. Publish the pose message.
7. Return the resized and annotated segmentation image.

To detect all the objects in the environment, the robot is moved around the room while continuously capturing RGB and depth images from its camera. These images are processed by the panoptic segmentation node, which detects objects, assigns labels, and estimates 3D positions. However, not all detections are recorded. Object poses are only considered when the robot is standing still. This is to avoid errors caused by motion blur or localization drift during movement. When the robot stops, the assumption is that the localization is more accurate, and the detection results are more reliable.

Each time the robot stops, the segmentation node runs and outputs a list of detected objects. For each object, the center of its bounding box is used to extract a depth value from the depth image. This depth, combined with the camera intrinsics, is used to calculate the 3D position of the object in the camera frame. The result is then transformed to the map frame using the robot's current pose estimate. The output is published as a message that includes the object labels and their corresponding 3D poses.

The task of checking for duplicate objects is handled later in the semantic mapping node. To avoid recording the same object multiple times from different angles, the system compares new detections with previously stored ones of the same label. If a new detection is within a defined distance threshold (ϵ) of an existing object, it is treated as the same object and skipped. This filtering helps maintain a clean and consistent map, even as the robot views objects from various perspectives.

Over time, as the robot explores the space and stops at different locations, a fuller and more complete map of the environment is built.

The final output of the detection pipeline consists of:

- A list of 3D poses representing the detected object positions in the map frame.
- A corresponding list of object labels (e.g., "chair," "table," "bottle") describing the detected objects.

These two lists are passed to the semantic mapping pipeline for clustering, categorization, and publishing of the semantic map.

5.1.2 Clustering objects

When the system has all the objects in the environment it needs to figure out which objects are connected to correctly infer the location. To do this a clustering algorithm is implemented, as mentioned in subsection 4.2.4 the choice of clustering algorithm is a version of

DBScan which takes into account the map. It has the parameters ϵ which describes the max distance between points and minimum amount of data points in the cluster needs to be decided.

The steps this clustering algorithm makes is the following:

1. Go through each object pose and find the neighbours based on ϵ
2. For each neighbour check if there is an obstacle inbetween, remove any neighbors with an obstacle inbetween.
3. If there are less than *min_samples* label as noise, otherwise give it cluster id *cid*
4. Go through seeds which are the neighbours to the start object pose
5. For each seed, check for objects which has not been visited and within ϵ distance of the seed.
6. Repeat step 2-5 until all poses have been visited.
7. Go through the list of poses and find entries which has the same *cid*, append object pose and object label to separate cluster and cluster label lists.
8. Append cluster and cluster label lists to a list containing all clusters and a list containing all cluster labels.
9. Repeat for all clusters
10. Return the list of all clusters and the list containing all cluster labels.

All of these steps are described in the pseudo code described in algorithm 2.

Algorithm 2 Map-Aware DBSCAN

Require: *poses, object_labels, ϵ , min_samples, optional map*

Ensure: clusters of poses and their labels

```
1: initialize all points as unvisited, label = unclassified,  $cid \leftarrow 0$ 
2: for all point  $i$  in poses do
3:   if not visited[ $i$ ] then
4:     visited[ $i$ ]  $\leftarrow$  true
5:      $n \leftarrow$  REGIONQUERY( $i, \epsilon, map$ )  $\triangleright$  Returns a list of all the neighbours
6:     if  $|n| < min\_samples$  then
7:       label[ $i$ ]  $\leftarrow$  noise
8:     else
9:       label[ $i$ ]  $\leftarrow cid$ 
10:      seeds  $\leftarrow n$ 
11:      while seeds not empty do
12:        current  $\leftarrow$  seeds.pop()
13:        if not visited[current] then
14:          visited[current]  $\leftarrow$  true
15:           $m \leftarrow$  REGIONQUERY(current,  $\epsilon, map$ )
16:          if  $|m| \geq min\_samples$  then
17:            add unseen elements of  $m$  to seeds
18:          end if
19:        end if
20:        if label[current]  $\neq$  noise then
21:          label[current]  $\leftarrow cid$ 
22:        end if
23:      end while
24:       $cid \leftarrow cid + 1$ 
25:    end if
26:  end if
27: end for
28: return clusters and object labels
```

The way this implementation of DBSCAN is augmented compared to the standard version is the ability to check the visibility between object poses. This is achieved through the algorithm called Bresenham's Line Algorithm which can be seen in Figure 5.2.

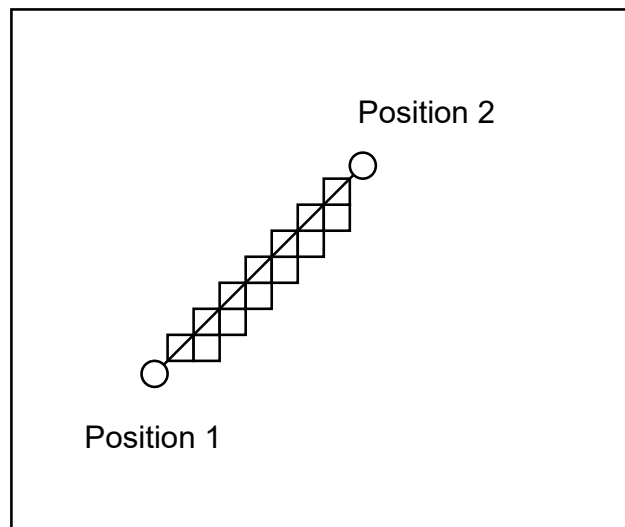


Figure 5.2: The Bresenham Line algorithm. It takes two position described with real world coordinates and computes the grid squares which form a line between the two points.

The purpose of this algorithm is to take the two poses, which in this case are the two object poses, and compute the line between them. The problem is the error between the poses needs to be essentially projected onto the grid which is what the algorithm 3 describes.

Algorithm 3 Check Line-of-Sight Between Two Grid Cells

Require: $(x_0, y_0), (x_1, y_1)$ ▷ Grid cell coordinates
1: $width, height$ ▷ Grid dimensions
2: $grid_data$ ▷ Flattened array of size $width \times height$
3: $wall_threshold$ ▷ Value indicating obstacle
Ensure: **true** if clear line-of-sight; **false** otherwise
4: **function** BRESENHAM_LINE(x_0, y_0, x_1, y_1)
5: $dx \leftarrow |x_1 - x_0|, \quad dy \leftarrow |y_1 - y_0|$
6: $sx \leftarrow (x_0 < x_1) ? +1 : -1$
7: $sy \leftarrow (y_0 < y_1) ? +1 : -1$
8: $err \leftarrow dx - dy$
9: $x \leftarrow x_0, \quad y \leftarrow y_0$
10: $points \leftarrow []$
11: **while** true **do**
12: append (x, y) to $points$
13: **if** $x = x_1$ **and** $y = y_1$ **then break**
14: **end if**
15: $e2 \leftarrow 2 \times err$
16: **if** $e2 > -dy$ **then**
17: $err \leftarrow err - dy; \quad x \leftarrow x + sx$
18: **end if**
19: **if** $e2 < dx$ **then**
20: $err \leftarrow err + dx; \quad y \leftarrow y + sy$
21: **end if**
22: **end while**
23: **return** $points$
24: **end function**

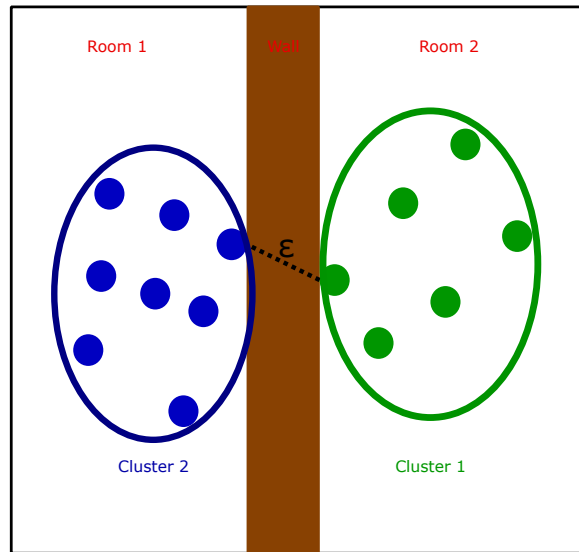


Figure 5.3: Clustering using the map aware DBSCAN described in Algorithm [2], object poses are recorded in two adjacent rooms and clustered. Objects in the two separate rooms does not end up in the same cluster since the algorithm checks whether there is a wall between them.

As can be seen in Figure 5.3 there are two sets of clusters in each room, if a standard DBSCAN algorithm would be used in this case the two clusters would end up in the same cluster which is not desired. Incorrect clusterings potentially leads to wrong location estimations based on the object labels. To fix this issue the clustering algorithm needs to account for the wall between objects. This can be done by passing in the map in the algorithm and use information such as the resolution of the map to link specific cells to the specific poses and computing the Bresenham line between two points as previously described. By making the algorithm aware of the obstacle between objects in the two rooms incorrect location estimations are avoided. Then the task is simply when adding another pose to the cluster, it checks whether the closest node object is to a wall which separates into a new cluster.

5.1.3 Location area estimation

As described in subsection 4.2.4 the area is computed using a MBR, the specific implementation does not find the orientation of the rectangle which provides the absolute smallest area.

Algorithm 4 Minimum Bounding rectangle

Require: Cluster of 2D points $C = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

Ensure: Width, height, and center of the bounding box

- 1: $x_{\min} \leftarrow \min(x_i \mid (x_i, y_i) \in C)$
 - 2: $x_{\max} \leftarrow \max(x_i \mid (x_i, y_i) \in C)$
 - 3: $y_{\min} \leftarrow \min(y_i \mid (x_i, y_i) \in C)$
 - 4: $y_{\max} \leftarrow \max(y_i \mid (x_i, y_i) \in C)$
 - 5: $width \leftarrow x_{\max} - x_{\min}$
 - 6: $height \leftarrow y_{\max} - y_{\min}$
 - 7: $center_x \leftarrow \frac{x_{\min} + x_{\max}}{2}$
 - 8: $center_y \leftarrow \frac{y_{\min} + y_{\max}}{2}$
 - 9: **return** ($width, height, center_x, center_y$)
-

As seen in Algorithm [4] the input to the location area is simply a cluster of object poses. The goal of algorithm is the find the width, height and the centre coordinates of the area. To do this it simply find the minimum and maximum points in the x and y-axis of the clusters. Once the maximum and minimum values have been found by simply finding the width and height to difference between the max and min on the different axes, and lastly finding the centre position by finding the average of the two. This is repeated for every cluster found in the object layer, each location is then saved in a JSON file as the semantic map, which can then be reconstructed by the semantic map server.

5.1.4 Location inference

This section will cover the location inference performed by the owl graph node. The knowledge graph is built in python using the OWL framework owlready2 with the structure described in Figure 4.4. When the node receives the service request to infer the location it can either receive an input of a list of object labels in the context of semantic mapping or a specific object or need in the context of navigating to specific locations. In the context of semantic mapping the node processes each object label one at a time where it essentially looks up in the built knowledge graph for instances where the label is the same. Based on which locations the objects are defined to be in it increments a counter for each location. As an example, if the node receives the label "chair" it can infer from the knowledge graph that the chair can be both in office and kitchen but not in the bedroom. Therefore the counters for the kitchen and office are incremented while the bedroom count remains the same. If the next object label is fridge the knowledge graph infers that the location can only be the kitchen and therefore the kitchen counter is incremented. If the object label is not defined in the knowledge graph the node will not increment anything. Once all the object labels in the input has been processed a label is returned based on which location counter is the largest. The described algorithm can be seen in algorithm 5 and can be used for both semantic mapping and context aware navigation.

Algorithm 5 Ontology-Based Location Inference

Require: List of input labels $E = [e_1, e_2, \dots, e_n]$

Ensure: Most likely inferred location or fallback message

```
1: Initialize dictionary  $L \leftarrow$  ▷ Counts of inferred locations  
   ▷ — First pass: match Objects directly —  
2: for all  $e \in E$  do  
3:   for all object  $o \in \text{Object.instances}()$  do  
4:     if  $o.name$  equals  $e$  then  
5:       for all location  $loc \in o.R$  do  
6:          $L[loc.name] \leftarrow L[loc.name] + 1$   
7:       end for  
8:     end if  
9:   end for  
10: end for  
11: if  $L$  is not empty then  
12:   return location with highest count in  $L$   
13: end if ▷ — Fallback: match Needs, then find Objects that fulfill them —  
14: Initialize list  $N \leftarrow []$  ▷ Matched needs  
15: for all  $e \in E$  do  
16:   for all need  $n \in \text{Need.instances}()$  do  
17:     if  $n.name$  equals  $e$  then  
18:       Append  $n$  to  $N$   
19:     end if  
20:   end for  
21: end for  
22: for all  $n \in N$  do  
23:   for all object  $o \in \text{Object.instances}()$  do  
24:     if  $n \in o.fulfills$  then  
25:       for all location  $loc \in o.R$  do  
26:          $L[loc.name] \leftarrow L[loc.name] + 1$   
27:       end for  
28:     end if  
29:   end for  
30: end for  
31: if  $L$  is not empty then  
32:   return location with highest count in  $L$   
33: end if  
34: return "No locations inferred."
```

5.2 Context aware navigation

In the context aware navigation part of the system the main component is the NLP Node which connects everything together.

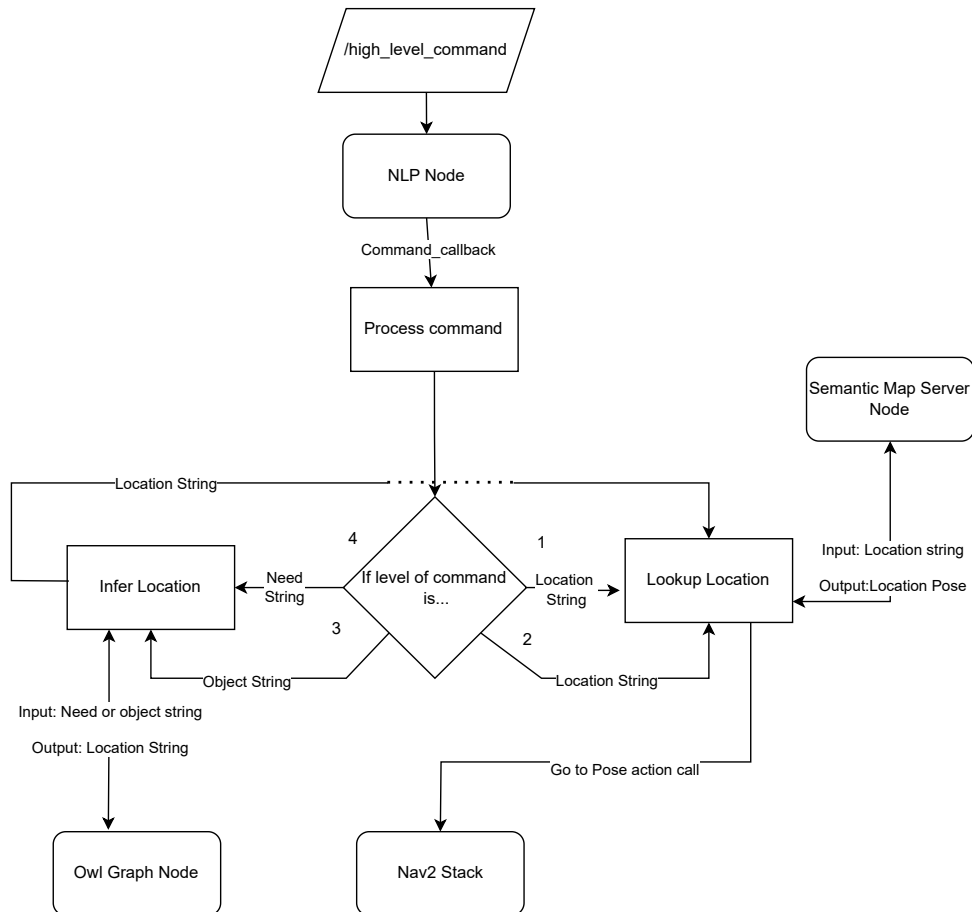


Figure 5.4: Context aware navigation flowchart. The NLP node subscribes to the high level command topic, the callback processes the command and depending on the level of action lookup the location from the map server. If the command is level 3 or 4 it first infers the locations first before looking up the location. Once it has the location pose it send a go to pose action call to the NAV2 stack.

As can be seen in the Figure 5.4 the operator starts the context aware navigation through the topic high level command topic. The command is processed and identified as one of the four commands described in Table 2.3. Depending on the level of the command it makes calls to the map server first or the owl graph node for the location string and then the map server. Once the location pose has been retrieved from the map server it send an

go to pose action call to the NAV2 stack.

5.2.1 NLP node

As described in 4.3.2, the robot must interpret high level commands from the user and turn them into navigation commands for the robot. To achieve this, an NLP node is implemented which makes use of the NLP framework spacy for interpretation. As described in section 3.2 this node needs to be able to handle up to level 4 commands, therefore the node needs a method to identify the expected keywords in each level of command. The spacy framework provide different attributes for each word in the given input, this will be the basis for how the keywords in different levels are found.

Level 1

As described in section 2.4 this command will simply be telling the robot to go to a specific location. The different components the node looks for at this level is simply an action and a location. To find the action the node looks for words which are verbs, this is words such as "go, navigate, move" etc. For the location the node looks for a noun, for the location specifically however the node keeps a table of the possible locations it can go to. Once a noun has been found the node checks whether it is part of the known location list, if found it can then send a request to the map server for the navigation pose. The reason the node has to keep a table of the possible locations is apparent when trying to process the level 2 command.

Level 2

For the level 2 command there are three keywords the node looks for, the action, the location and the object. The problem here is that both the object and the location will show as nouns which is why it needs to have the table of known locations. This way the node is able to recognize the difference between locations and objects.

Level 3

For the level three the procedure is almost the same as level 1, in this case the node looks for an action and a object. For this command the system needs to infer the location to navigate to through the knowledge graph.

Level 4

For the level 4 command the robot needs to be able to establish a need from the command and infer an object and a location from that specific need. While needs are often described using an adjective such "hungry, thirsty" etc. This is not the only way to describe a need, it can also be described as essentially wanting to perform a specific action. Consider the

following input "I want to cut a piece of paper". Humans will understand this command as they need something which can cut a piece of paper but there are not adjectives in this command so there are two methods which needs to be used to make sure the node capture as many needs as possible. The way to find a need in this format is to look at the verb and nouns again, the most common structure of the sentence will be a verb describing the action the user wants to take and the object they want to perform the action on. Therefore the node looks specifically for a verb followed by a noun and uses the base form of the verb as the specific need keyword.

Command Parsing

As can be seen in algorithm 6 the node processes the same input through all four of the different levels. Essentially what each of levels do is they take the input and check for the previously described keywords expected in each level. If the keywords are not present in the input it simply returns an indication that there is no output for that specific level. In the case of the level 2 input there are some complications since the expected keywords of a level 1 and level 3 are subsets of the expected keywords of level 2. Therefore if the command has the necessary keywords to classify it as a level 2 command it also qualifies for level 1 and 3. To solve this issue A slight change in the ordering of the commands was implemented. Essentially the first level being checked is level 2, if it has the necessary components the command is then processed.

Algorithm 6 Analyze Command Across Four Levels

Require: input command text T

Ensure: predicted level ℓ and output string R

```
1:  $A \leftarrow \text{extract\_action}(T)$ 
2:  $L \leftarrow \text{extract\_location}(T)$ 
3:  $O \leftarrow \text{extract\_object}(T)$ 
4:  $N \leftarrow \text{extract\_need}(T)$ 
5:  $\ell, R \leftarrow \text{None}, \text{"Unable to parse command"}$   $\triangleright$  Try each parsing level in priority order
   (2,1,3,4)
6: if  $A \neq \emptyset \wedge L \neq \emptyset \wedge O \neq \emptyset$  then
7:    $\ell \leftarrow 2$ 
8:    $R \leftarrow A \parallel ', ' \parallel L \parallel ', ' \parallel O$ 
9: else if  $A \neq \emptyset \wedge L \neq \emptyset$  then
10:   $\ell \leftarrow 1$ 
11:   $R \leftarrow A \parallel ', ' \parallel L$ 
12: else if  $O \neq \emptyset$  then
13:   $\ell \leftarrow 3$ 
14:   $R \leftarrow O$ 
15: else if  $N \neq \emptyset$  then
16:   $\ell \leftarrow 4$ 
17:   $R \leftarrow \text{"Recognized need: " } \parallel N$ 
18: end if
19: return  $(\ell, R)$ 
```

5.2.2 Navigation

To run this section of navigation, three things happen first: a JSON file is outputted from semantic mapping pipeline which contains all the known locations and their coordinates. When the NLP node receives the location pose from the location lookup it sends it to Navigation2 (NAV2) in order to move the Tiago base robot from its current pose to location. The NAV2 is a version of the ROS 1 move_base stack, which enables the mobile robot to navigate in an environment autonomously. NAV2 allows the robot to:

- Plan a path to a goal pose (x, y, θ)
- Dynamically avoid obstacles
- utilize the maps for localization and path planning
- Perform smooth motion control to follow the path

6 Test setups and results

This chapter aims to test the different components of the system. It will be split up into a section talking about the test setups and a section which presents the results of the test. A link to videos of the testing can be found in the preface.

6.1 Test Setups

This section describes the test setups, first there will be component tests which evaluates the performance of the different components of the pipeline. Lastly will be system test where the performance of the system on the use case described in section 2.4.

6.1.1 Test 1: Object Detection

For semantic mapping to work properly it requires the object detection to work properly as a wrong label will lead to a wrong estimation of the location. In addition to this the pose estimation will also be evaluated, more accurate pose estimation of the objects will give a more accurate location estimation.

Goal of the test

1. Evaluate the accuracy of the object detection

Test Setup

The environments are setup such that there are a few different objects at each location, the robot should then move around at each location to find as many of the present objects.

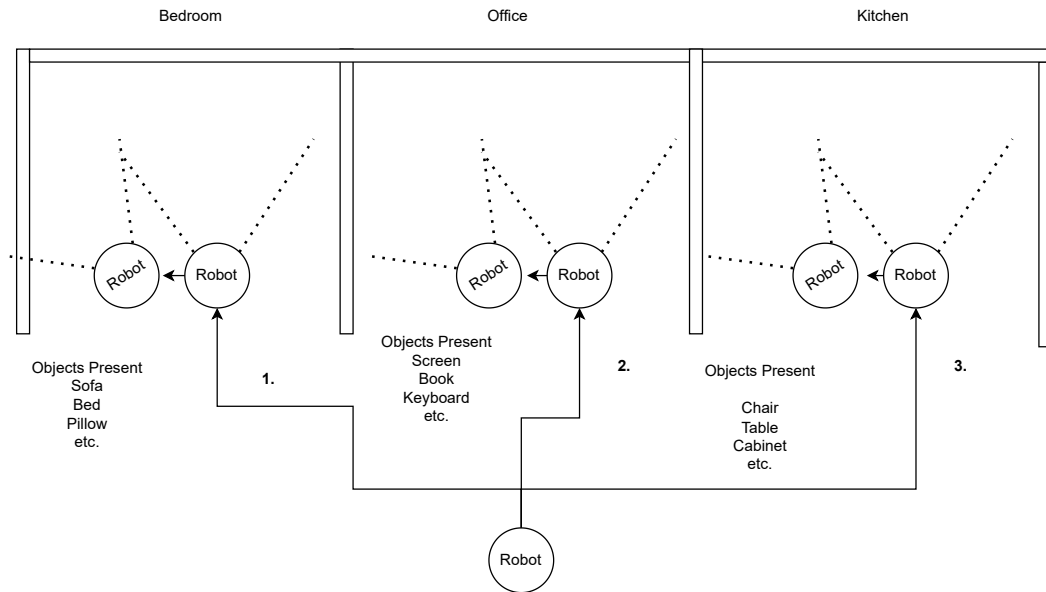


Figure 6.1: Test setup for the object detection test. There are the three locations which have different objects, the robot will move to each of these locations and find as many of the present objects.

Protocol

The robot is loaded into the prebuilt environment where the position and objects are known.

1. Place objects in the simulation environment
2. Make a list of objects present in each location
3. Start the object detection node
4. Move the robot around the location
5. Record detected object labels
6. When the camera have been placed at locations such that all objects had the opportunity to be seen move to the next location.
7. Repeat until all locations have been covered

Evaluation

Before the test is made a ground truth list of the objects are recorded, this is based on the location in the simulated environment. Once this has been established and a list of detected

object have been made it will be compared with the ground truth. Each data point can have one of three values, either it successfully found the object, it missed the object or a false positive which means it found a label which was not defined in the ground truth. The success of the test will be based on the results for each location.

6.1.2 Test 2: Keyword Extraction

The keyword extraction mainly uses the transformer model BERT. To work properly, it requires the Tiago base robot to understand the inputs of the user. First, it needs to understand the action and then understand the target for the action. When the user gives the robot **"fetch me a glass of water"**, in this case, the action would be *"fetch"* and the target action would be *"glass of water"*.

Goal of the test

1. Evaluate the NLP node's ability to extract the correct keywords.

Test Setup

25 commands for each level is prepared along with a list with the expected keywords the NLP node should output. Then each command is processed and compared with the expected value.

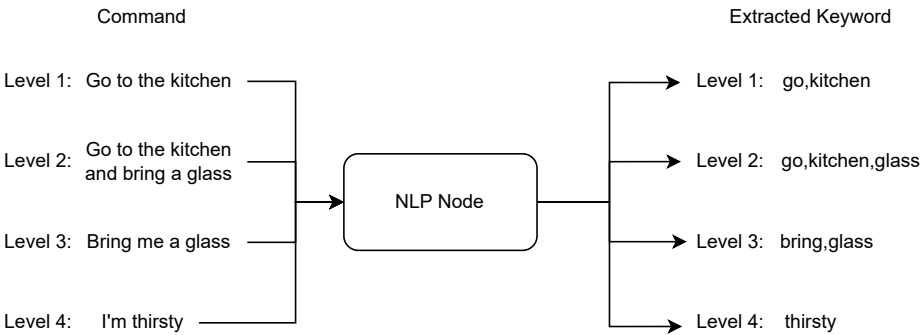


Figure 6.2: Test 2 setup. A list for each level of command is made and processed by the NLP node. This outputs keywords which are different for each level. These keywords are compared with a ground truth and estimated to be correct or wrong.

Protocol

The robot is loaded into the prebuilt environment where the position and objects are known.

1. 25 commands for each level is prepared, 100 in total along the expected keywords for each command.
2. Input a command and process it
3. Compare the outputted keywords with the expected keywords
4. Repeat 2 and 3 for all prompts in each level

Evaluation

The test will be evaluated based on the amount of correct keywords the algorithm extracts from the different level of commands up to level 4.

6.1.3 Test 3: Location Inference

The location inference looks into how ontology utilizes keyword extraction in order to build the Knowledge graph so the robot can understand the context and actions which been given and made into instructions.

Goal of the test

1. Evaluate the accuracy of the location inference

Test Setup

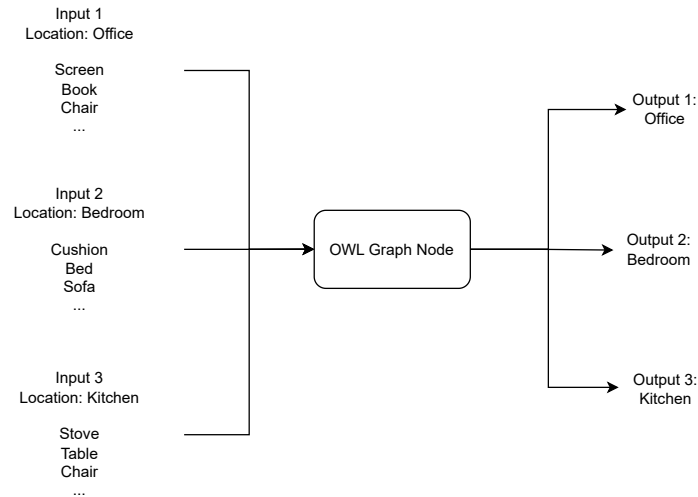


Figure 6.3: Test 3 Setup. There are three different inputs for each of the locations, where an input is a list of objects expected to be in that environment. The input is processed by the OWL graph node which outputs a location based on the objects in the input list.

Protocol

1. Start up the simulation
2. Move the robot around the different rooms
3. Record the object labels
4. Cluster the objects
5. Infer a location based on the object labels in a cluster
6. Compare inferred location with expected location

Evaluation

The success of this test will be based on whether it is able to correctly infer the locations based on the objects seen in the environment. The inferred location will be compared with the ground truth, if all the inferred locations are the same as the ground truth the test will be considered a success.

6.1.4 Test 4: Semantic mapping

The semantic mapping node uses all the data from the different nodes to assemble into the semantic map. Which receives record all inputs from object detection node, locate the objects in the global map frame, and then it cluster the objects into specific locations, Compute the areas covered by the clusters and at the end Infer the location based on object labels.

Goal of the test

1. Evaluate the accuracy of the Object Location relations
2. Evaluate the accuracy of the Localization in the existing map
3. Evaluate the accuracy of the Object Labels and relative position

Test Setup

The robot will move around the simulation while calling on the semantic mapping node to identify the object and its location of the object. Then look into the NAV2 for the localization in the existing map. In the end object labels and relative position will be tested.

Protocol

1. Start up the simulation
2. Move the robot around the different rooms
3. Record the object labels
4. Cluster the objects
5. Infer a location based on the object labels in a cluster
6. Compare location with ground truth
7. Compare location Object Labels and relative position

6.1.5 Test 5: Command execution

This test of command execution would mainly look into how the full system execute the given action through using keyword extraction and ontology.

Goal of the test

1. Evaluate the full systems ability to handle the different levels of comamnds.

Test Setup

The robot will be placed at any position in the simulation environment, from this point the robot needs to be able to move to each of the locations using all four levels of commands.

Protocol

1. Start up the simulation.
2. Load the semantic map into the semantic map server.
3. Give the system a level 1-4 command for each room.
4. Note the end pose of the robot at the end of the navigation.

Evaluation

The robot will be evaluated on whether it reaches the desired location, it does not have to specifically be the goal pose retrieved from the semantic map server but simply somewhere within the estimated location area.

6.2 Test Results

This section will cover the test results, data collected according to the protocol described in section 6.1 will be presented here and an evaluation of the test will be presented.

6.2.1 Test 1: Object Detection

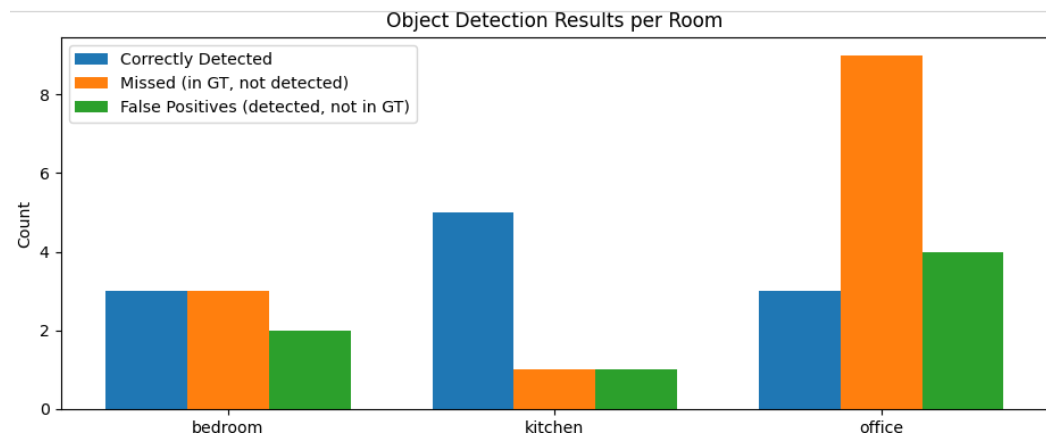


Figure 6.4: Object detection test results. Three different things were collected here, first the number of correct object detections compared with the pre established set of objects in each locations, this is represented by the blue color. The orange color describes the number of objects which were present in each location but not detected by the object detection. Lastly the green color represents objects detected which were not established to be in each location.

As can be seen in Figure 6.4 the object detections had a very inconsistent performance. The most success were found in the kitchen area where only a single object were not seen and a single false positive out of the seven different objects described to being in the environment. The location where the object detection were least successful were the office locations where the majority of the different objects were not detected and about four false positives. Lastly the bedroom had a 50/50 split between correctly detected and missed objects with about 2 false positives.

6.2.2 Test 2: Keyword Extraction

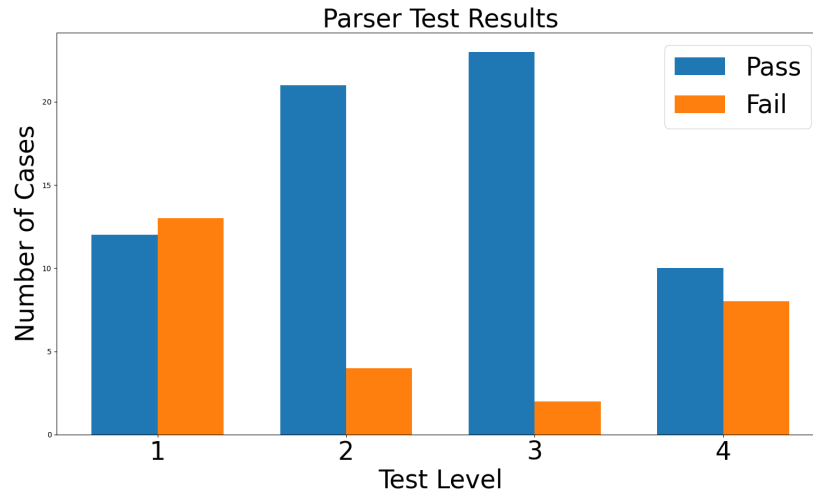


Figure 6.5: The results for test 2

As can be seen in Figure 6.5 the system does a good job of detecting the different kinds of commands the user could give. Specifically level 2 and 3 the system performed well with about 84% and 92% accuracy respectively as can be seen in Table 6.1. Level 1 and 4 did not get the best results with 48% and 55% respectively.

Level	Pass	Fail	Accuracy
1	12	13	48%
2	21	4	84%
3	23	2	92%
4	10	8	55%

Table 6.1: Test 2 results

6.2.3 Test 3: Location Inference

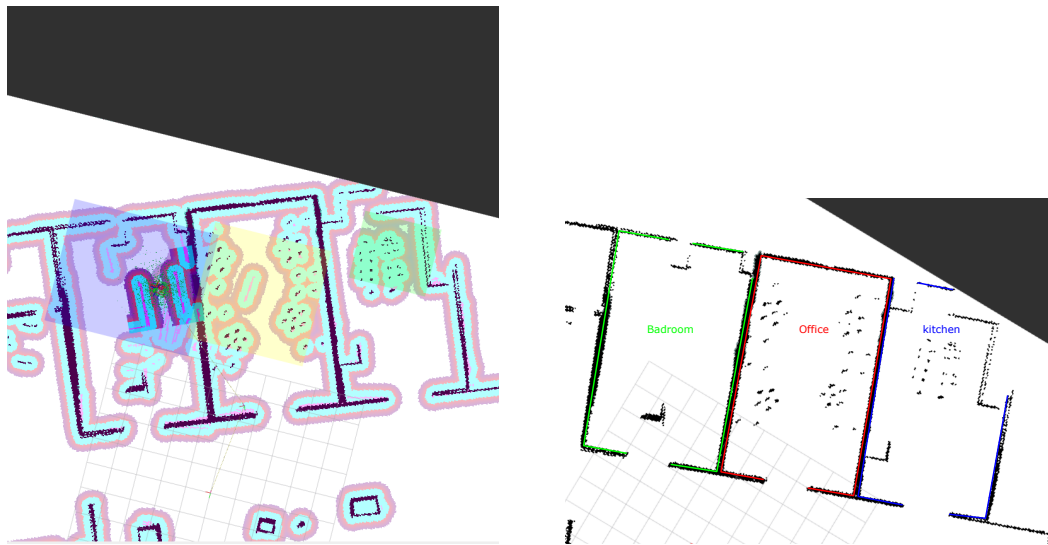
Based on the objects found in test 1 it can infer the location based on objects in the environment It also successfully infers location based on specific needs.

Test	Object labels	Inferred	Expected
1	tv stand, trashcan, sofa, bedroom, pillow, lamp	Bedroom	Bedroom
2	refrigerator, sink, cabinet, chair, table, can,	Kitchen	Kitchen
3	screen, keyboard, mouse, mousepad, speaker, book, paper, bowl, cup, pen, swivel chair, phone, desk,	Office	Office

Table 6.2: Results of test 3. The location inference was able to correctly estimate all three locations with the given object labels.

As can be seen in Table 6.2 using the objects established in test 1 it finds bedroom, kitchen and office correctly.

6.2.4 Test 4: Semantic Mapping



(a) The output of the semantic mapping algorithm. The algorithm found the three locations, bedroom is blue, yellow is the office and green is the kitchen.

(b) The ground truth mapping of the locations.

Figure 6.6: Comparison between the output of the semantic mapping pipeline (a) and the established ground truth placement of the locations (b).

As can be seen in Figure 6.6 the semantic mapping output was able to be fairly accurate in the mapping of the locations when compared with the ground truth defined in Figure 6.6 (b). While the locations are placed at the correct locations they do not quite fill out the

room as depicted in Figure 6.6. This is somewhat expected, as while humans understand it as the whole room is the location, the semantic mapping method will only try to find the area which contains all the detected objects. The clustering did well in separating the different rooms however there are some issues with this implementation. Consider the kitchen area where there is a countertop with a sink in it. While the object detections could see the sink and locate it, the pose of the sink would be behind the mapped wall, which in reality is the countertop. So when the algorithm tried to add the sink to the kitchen cluster, it simply were excluded due to the line between the closet cluster object and the sink would not be clear of obstacles. Overall the semantic mapping test is considered a success, but there are a few points which could be addressed.

6.2.5 Test 5: Command Execution

This was the full system test, as long as the input were correctly processed it was able to handle the four different levels. One occasional issue it had was the navigation goal for each location. Since we're currently just going for the center of the location there are times where the navigation goal would be inside an object.

	Level 1	Level 2	Level 3	Level 4
Office	✓	✓	✓	✓
Kitchen	✓	✓	✓	✓
Bedroom	✓	✓	✓	✓

Table 6.3: Test 5 results. As can be seen the the system were able to navigate to the three different locations through the use of the different levels of command.

As can be seen in Table 6.3 the robot was able to navigate to all the different locations using the different levels of command. There were specific locations however where the robot had an easier time navigating to, specifically the office location where much more accessible for the robot as there were nothing placed in the middle of the location area. The bedroom and kitchen however were more trouble some as objects and obstacles were placed near the centre of the location which means occasionally the robot had a hard time moving to the exact spot it wanted. While all test are considered successful this is in the sense that it always ended the navigation in the room.

7 Discussion

This chapter will cover the discussion of the project, it will reflect over the test results for the different components and the overall system tests.

7.1 Test results

This section will cover the discussion of the test results. This covers the reflection over the result, where it succeeded and where each component fell short and potential reasons why it fell short.

Object detection

While the object detection did not have the best results with regards to detection objects correctly the system were still able to perform correctly. But why is it the case that it did not perform so well in the test. The first reason could be the simulation environment, while there it was attempted to recreate a real environment there still was a distinct lack of real world features. Specifically, there was a lack of colour and generally different colours in the environment and of the different objects, respectively. The office location which had the largest amount of missed objects also had the largest amount of smaller object, such as phones, cups, books, etc. These smaller object can be particularly hard to detect, especially in a featureless environment. There were also a number of cases where the object detection fuses multiple objects into a single object, an example of this is the object "computer". A computer have many different components which add up to the full system, this includes objects such as a screen, keyboard and mouse, all objects which were included in the ground truth of the objects it could detect. These objects, however, were often fused into a single object of a computer, which makes the results of the test indicate that it missed more objects than it really did.

Other reasons for the model mislabelling or not detecting the object in the environment are highly likely due to the dataset used, which was the large base COCO dataset. This COCO dataset, even if it has similar label objects to the ones used in the simulation environment, does not have the exact same objects in terms of colour, texture, length, height, etc. This resulted in mislabelling or completely missing the object because the model could not recognize it due to different features, even though the model had a label for that object.

Natural Language processing

While the NLP component did well on the test there are still alot of issues which needs to be addressed. First of all the way the logic is built leaves very little room for ambiguity

in the command inputs it can take. Of the shortcut that had to be made in it was first to distinct between objects and locations. Since a locations such as kitchen and object such as glass are both nouns it runs into the issue of both being able to be labelled as an object. To avoid this happening the NLP module was augmented with a list of possible locations which would be used when checking if a noun was a location. This adds more configuration and issues with scalability which the system already suffers from. In addition to that workaround it also had issues with ambiguous inputs in level 1 and 4 commands, specifically different ways to describe the action. Currently it relies on a verb being used to describe the action to take, however in some cases humans would use an input like "head into the kitchen" where the action is described used nouns.

Knowledge graph

While the knowledge graph test were successful there are some more underlying issues with this part of the implementation, specifically in scalability. Adding new objects or relations is creates exponentially more setup for the pipeline to work in different environments. If the environments and relations between objects in those environment remain small in size this implementation of the knowledge graph should be fine but if more complex tasks needs to be solved another method should be considered.

Semantic mapping

Overall the semantic mapping pipeline fulfilled the base requirements of being able to find identify the locations and their placement in the environment. There still are some a few workarounds which could be improved upon as mentioned in subsection 6.2.4. The current implementation requires the robot to stand still to record the poses of the objects in the environment which differers from other mapping algorithms such as the conventional SLAM implementations where the mapping happens more seamlessly.

Context aware navigation

While the system was able to perform the context aware navigation using the different levels of commands there are still some navigational issues which has not been account for. The current implementation still only receives the centre coordinates of each area which could potentially result in issues for the robot. Consider the kitchen area where the dinner table is often placed in the middle of the room, this results in when the robot retrieves the pose for the kitchen area it would evidently try to navigate into the middle of the table resulting in a failed go to pose task. Also the issue of finding the objects have not really been explored, this means when the robot reaches the location if the object is not in sight of the robot it will not be able to find a local goal pose to move into range of the grasping.

7.2 Future work

This section will cover potential future works in the project, this could be changes to improve the performance in areas not covered in this report.

Object detection

There are a few things that could be improved in future work. First, using a more realistic simulation environment with better lighting, more textures, and more variation in colours could help the model detect objects more accurately. The current simulation lacked many of the real-world features that would normally help object detection. Second, using a custom dataset that matches the simulation objects more closely would likely improve results. The COCO dataset has similar labels, but the actual objects look different in terms of size, shape, colour, and texture. Training the model on examples that look more like the ones in the environment would help it recognize them better. Lastly, better handling of compound objects like computers (which include screens, keyboards, and mice) could improve the accuracy of the detection results. Right now, the model tends to fuse these into a single object, which makes the test results look worse than they actually are. One solution could be to train the model to detect the smaller parts individually and then group them logically afterward.

Natural language processing and knowledge graph

While the NLP module and knowledge graph was successful in completing the test made in this project there still is a question of scalability. As described in subsection 5.1.4, to build the knowledge graph every single object, location, need and relation between different object had to be manually defined. While this approach works for simple object relations and a limited scope on the environments, more locations, objects and even a larger semantic understanding of the environment can quickly get out of control.

The same can be said for the NLP component, it did a good job on the the different levels of commands, in the same manner as the knowledge graph the scalability

Semantic mapping

For the semantic mapping part, there are still improvements that could be made to the current pipeline. Right now, the robot has to be standing still to record the object poses, which limits how flexible the mapping process is. Future work could focus on making the system work more like traditional SLAM methods, where mapping can happen while the robot is moving. This would make the process faster and more efficient. Some of the workarounds mentioned earlier could also be improved or removed with a more advanced mapping system. The current location estimation could be slightly improved as well, the current implementation does not account for the orientation of the area which lead to a

slight difference between the estimated ground truth and the output of the location area. The choice of MBR as the location estimate still seems like the optimal choice for both storing the location and as the estimated area. It could potentially lead to issues if the room is not rectangular but this is an edge case which should not be a common occurrence.

Context aware navigation

As for context aware navigation, one of the main problems is that the robot only receives the centre pose of each area. This can cause navigation issues, especially in rooms with big objects in the middle, like kitchen tables. In these cases, the robot might try to go straight into the table, which causes it to fail the task. A better system would be to provide the robot with a set of valid poses or an area to move within, rather than just a single point. Also, object search and discovery could be improved. Right now, if the robot reaches a location and can't see the object, it won't try to search for it or adjust its position. Adding a local search strategy to find objects that are not in direct view would make the system more reliable in real-world use.

8 Conclusion

The goal of this project was to develop a context-aware navigation system that allows a service robot to perform navigation tasks based on human input. This was achieved by building a pipeline that integrates semantic mapping, natural language processing, and a navigation stack. The semantic mapping was able to detect and localize objects in the environment, cluster them into areas, and assign scene labels using ontology. The NLP node successfully extracted commands and translated them into actions and targets, which were then used by the navigation system to guide the robot.

The system was tested in simulation and was able to complete several levels of commands, such as moving to specific rooms or searching for objects. However, some limitations were found during testing. For example, the robot could only record object positions while standing still, and navigation relied on static center coordinates, which sometimes caused collisions with furniture. Also, object search behaviour was not included if the target was not in direct view.

Overall, the system showed that it is possible to use human context and semantic information to guide a robot's actions in a structured environment. The results indicate that this approach can be extended with improvements such as continuous object mapping, smarter pose selection, and more flexible object search strategies. These areas would help make the system more robust and closer to being used in real-world service robotics applications.

Bibliography

- [1] European Parliament et al. *Industry 4.0*. European Parliament, 2016. doi: [doi/10.2861/947880](https://doi.org/10.2861/947880).
- [2] European Commission et al. *Industry 5.0, a transformative vision for Europe – Governing systemic transformations towards a sustainable industry*. Publications Office of the European Union, 2021. doi: [doi/10.2777/17322](https://doi.org/10.2777/17322).
- [3] Grace Titilayo Babalola et al. “A systematic review of collaborative robots for nurses: where are we now, and where is the evidence?” In: *Frontiers in Robotics and AI* 11 (2024). ISSN: 2296-9144. doi: [10.3389/frobt.2024.1398140](https://doi.org/10.3389/frobt.2024.1398140). URL: <https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2024.1398140>.
- [4] International Federation of Robotics. *Service Robots*. URL: <https://ifr.org/service-robots/>. (accessed: 21.03.2025).
- [5] Amit Kumar Pandey and Rodolphe Gelin. “A Mass-Produced Sociable Humanoid Robot: Pepper: The First Machine of Its Kind”. In: *IEEE Robotics & Automation Magazine* 25.3 (2018), pp. 40–48. doi: [10.1109/MRA.2018.2833157](https://doi.org/10.1109/MRA.2018.2833157).
- [6] Jordi Pagès, Luca Marchionni, and Francesco Ferro. “TIAGo: the modular robot that adapts to different research needs”. In: 2016. URL: <https://api.semanticscholar.org/CorpusID:218478582>.
- [7] David Fischinger et al. “HOBBIT - The Mutual Care Robot”. In: Nov. 2013.
- [8] Melonee Wise et al. “Fetch Freight : Standard Platforms for Service Robot Applications”. In: 2016. URL: <https://api.semanticscholar.org/CorpusID:42886148>.
- [9] SoftBank Robotics. *Pepper*. URL: <https://us.softbankrobotics.com/pepper>. (accessed: 01.04.2025).
- [10] PAL Robotics. *TIAGo*. URL: <https://pal-robotics.com/robot/tiago/>. (accessed: 20.03.2025).
- [11] U.U. Samantha Rajapaksha, Chandimal Jayawardena, and Bruce A. MacDonald. “ROS Based Multiple Service Robots Control and Communication with High Level User Instruction with Ontology”. In: *2021 10th International Conference on Information and Automation for Sustainability (ICIAfS)*. 2021, pp. 381–386. doi: [10.1109/ICIAfS52090.2021.9606062](https://doi.org/10.1109/ICIAfS52090.2021.9606062).
- [12] Christian Tamantini et al. “A Robotic Assistant for Logistics and Disinfection in Health Centers”. In: Jan. 2021.
- [13] Markus Bajones et al. “Results of Field Trials with a Mobile Service Robot for Older Adults in 16 Private Households”. In: *J. Hum.-Robot Interact.* 9.2 (Dec. 2019). doi: [10.1145/3368554](https://doi.org/10.1145/3368554). URL: <https://doi.org/10.1145/3368554>.

- [14] Niko Sünderhauf et al. "Place categorization and semantic mapping on a mobile robot". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 5729–5736. DOI: [10.1109/ICRA.2016.7487796](https://doi.org/10.1109/ICRA.2016.7487796).
- [15] Denis F. Wolf and Gaurav S. Sukhatme. "Semantic Mapping Using Mobile Robots". In: *IEEE Transactions on Robotics* 24.2 (2008), pp. 245–258. DOI: [10.1109/TR0.2008.917001](https://doi.org/10.1109/TR0.2008.917001).
- [16] Mikel Galar et al. "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes". In: *Pattern Recognition* 44.8 (2011), pp. 1761–1776. ISSN: 0031-3203. DOI: <https://doi.org/10.1016/j.patcog.2011.01.017>. URL: <https://www.sciencedirect.com/science/article/pii/S0031320311000458>.
- [17] Ioannis Kostavelis and Antonios Gasteratos. "Semantic mapping for mobile robotics tasks: A survey". In: *Robotics and Autonomous Systems* 66 (2015), pp. 86–103. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2014.12.006>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889014003030>.
- [18] Zhiliu Yang and Chen Liu. "TUPPer-Map: Temporal and Unified Panoptic Perception for 3D Metric-Semantic Mapping". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 1094–1101. DOI: [10.1109/IROS51168.2021.9636599](https://doi.org/10.1109/IROS51168.2021.9636599).
- [19] Panqu Wang et al. *Understanding Convolution for Semantic Segmentation*. 2017. arXiv: [1702.08502](https://arxiv.org/abs/1702.08502) [cs.CV]. URL: <https://arxiv.org/pdf/1702.08502.pdf>.
- [20] Gabriela Csurka, Diane Larlus, and Florent Perronnin. "What is a good evaluation measure for semantic segmentation?" In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2013. URL: <https://projec.liris.cnrs.fr/imagine/pub/proceedings/BMVC-2013/Papers/paper0033/paper0033.pdf>.
- [21] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. "A survey on instance segmentation: state of the art". In: *International Journal of Multimedia Information Retrieval* 9.3 (2020), pp. 171–189. ISSN: 2192-662X. DOI: [10.1007/s13735-020-00195-x](https://doi.org/10.1007/s13735-020-00195-x). URL: <https://doi.org/10.1007/s13735-020-00195-x>.
- [22] Wenchao Gu, Shuang Bai, and Lingxing Kong. "A review on 2D instance segmentation based on deep neural networks". In: *Image and Vision Computing* 120 (2022), p. 104401. ISSN: 0262-8856. DOI: <https://doi.org/10.1016/j.imavis.2022.104401>. URL: <https://www.sciencedirect.com/science/article/pii/S0262885622000300>.
- [23] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. "Recurrent Instance Segmentation". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 312–329. ISBN: 978-3-319-46466-4.

- [24] Rohit Mohan and Abhinav Valada. “EfficientPS: Efficient Panoptic Segmentation”. In: *International Journal of Computer Vision* 129.5 (2021), pp. 1551–1579. ISSN: 1573-1405. DOI: [10.1007/s11263-021-01445-z](https://doi.org/10.1007/s11263-021-01445-z). URL: <https://doi.org/10.1007/s11263-021-01445-z>.
- [25] Alexander Kirillov et al. “Panoptic Segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.
- [26] Omar Elharrouss et al. “Panoptic Segmentation: A Review”. In: *CoRR* abs/2111.10250 (2021). arXiv: [2111.10250](https://arxiv.org/abs/2111.10250). URL: <https://arxiv.org/abs/2111.10250>.
- [27] Swapna. *Convolutional Neural Network | Deep Learning*. Accessed: 24-03-2025. 2020. URL: <https://developersbreach.com/convolution-neural-network-deep-learning/#1-what-is-cnn>.
- [28] Omar Elharrouss et al. “Backbones-review: Feature extractor networks for deep learning and deep reinforcement learning approaches in computer vision”. In: *Computer Science Review* 53 (Aug. 2024), p. 100645. ISSN: 1574-0137. DOI: [10.1016/j.cosrev.2024.100645](https://doi.org/10.1016/j.cosrev.2024.100645). URL: <http://dx.doi.org/10.1016/j.cosrev.2024.100645>.
- [29] Huiyu Wang et al. *MaX-DeepLab: End-to-End Panoptic Segmentation with Mask Transformers*. 2021. arXiv: [2012.00759](https://arxiv.org/abs/2012.00759) [cs.CV]. URL: <https://arxiv.org/abs/2012.00759>.
- [30] Bowen Cheng et al. *Masked-attention Mask Transformer for Universal Image Segmentation*. 2022. arXiv: [2112.01527](https://arxiv.org/abs/2112.01527) [cs.CV]. URL: <https://arxiv.org/abs/2112.01527>.
- [31] Zhiqi Li et al. *Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers*. 2022. arXiv: [2109.03814](https://arxiv.org/abs/2109.03814) [cs.CV]. URL: <https://arxiv.org/abs/2109.03814>.
- [32] Feng Li et al. *Mask DINO: Towards A Unified Transformer-based Framework for Object Detection and Segmentation*. 2022. arXiv: [2206.02777](https://arxiv.org/abs/2206.02777) [cs.CV]. URL: <https://arxiv.org/abs/2206.02777>.
- [33] François Chollet. “Deep Learning with Python”. In: Manning Publications Co., 2017, pp. 294–296.
- [34] Zhe Hu et al. “Safe Navigation With Human Instructions in Complex Scenes”. In: *IEEE Robotics and Automation Letters* 4 (2018), pp. 753–760. URL: <https://api.semanticscholar.org/CorpusID:52198622>.
- [35] Jesse Thomason et al. “Vision-and-Dialog Navigation”. eng. In: (2019).
- [36] Mohammad Taufik et al. “Voice-Controlled Interaction of Medical Robots Using Deep Learning with Gated Recurrent Units”. eng. In: *2024 IEEE International Conference on Control & Automation, Electronics, Robotics, Internet of Things, and Artificial Intelligence (CERIA)*. IEEE, 2024, pp. 1–6.
- [37] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: [1706.03762](https://arxiv.org/abs/1706.03762) [cs.CL]. URL: <https://arxiv.org/abs/1706.03762>.
- [38] W3C. *Web Ontology Language (OWL)*. URL: <https://www.w3.org/OWL/>. (accessed: 25.03.2025).

- [39] Luis Riazuelo et al. “RoboEarth Semantic Mapping: A Cloud Enabled Knowledge-Based Approach”. In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 432–443. DOI: [10.1109/TASE.2014.2377791](https://doi.org/10.1109/TASE.2014.2377791).
- [40] Yu Li et al. *Where to Fetch: Extracting Visual Scene Representation from Large Pre-Trained Models for Robotic Goal Navigation*. 2024. arXiv: [2408.10578](https://arxiv.org/abs/2408.10578) [cs.R0]. URL: <https://arxiv.org/abs/2408.10578>.
- [41] Yinpei Dai et al. *Think, Act, and Ask: Open-World Interactive Personalized Robot Navigation*. 2024. arXiv: [2310.07968](https://arxiv.org/abs/2310.07968) [cs.R0]. URL: <https://arxiv.org/abs/2310.07968>.
- [42] Haru Nakajima and Jun Miura. *Combining Ontological Knowledge and Large Language Model for User-Friendly Service Robots*. 2024. arXiv: [2410.16804](https://arxiv.org/abs/2410.16804) [cs.R0]. URL: <https://arxiv.org/abs/2410.16804>.
- [43] Ying Zhang, Guohui Tian, and Xuyang Shao. “Safe and Efficient Robot Manipulation: Task-Oriented Environment Modeling and Object Pose Estimation”. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–12. DOI: [10.1109/TIM.2021.3071222](https://doi.org/10.1109/TIM.2021.3071222).
- [44] Zhihao Li et al. “Intention Understanding in Human–Robot Interaction Based on Visual-NLP Semantics”. eng. In: *Frontiers in neurorobotics* 14 (2021), pp. 610139–610139. ISSN: 1662-5218.
- [45] UCL Course Notes. *Camera Calibration*. Accessed: 2025-06-01. 2024. URL: https://mphy0026.readthedocs.io/en/latest/calibration/camera_calibration.html.
- [46] Wikipedia contributors. *Camera Resectioning*. Accessed: 2025-06-01. 2024. URL: https://en.wikipedia.org/wiki/Camera_resectioning.
- [47] Karthikeya Shodhan. *What are Intrinsic and Extrinsic Camera Parameters in Computer Vision?* Accessed: 2025-06-01. 2022. URL: <https://towardsdatascience.com/what-are-intrinsic-and-extrinsic-camera-parameters-in-computer-vision-7071b72fb8ec/>.
- [48] Kyle Simek. *Intrinsic Parameters*. Accessed: 2025-06-01. 2013. URL: <https://ksimek.github.io/2013/08/13/intrinsic/>.

A Appendix

Here is the appendix

A.1 panoptic segmentation node

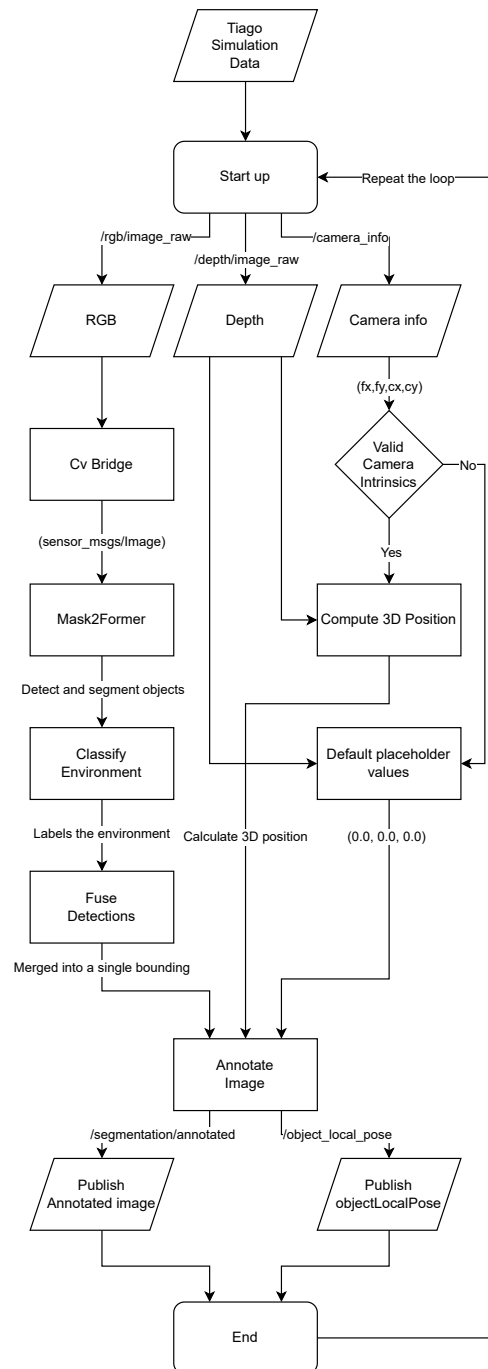


Figure A.1: Panoptic_segmentation_node.py

A.2 Simple dbscan

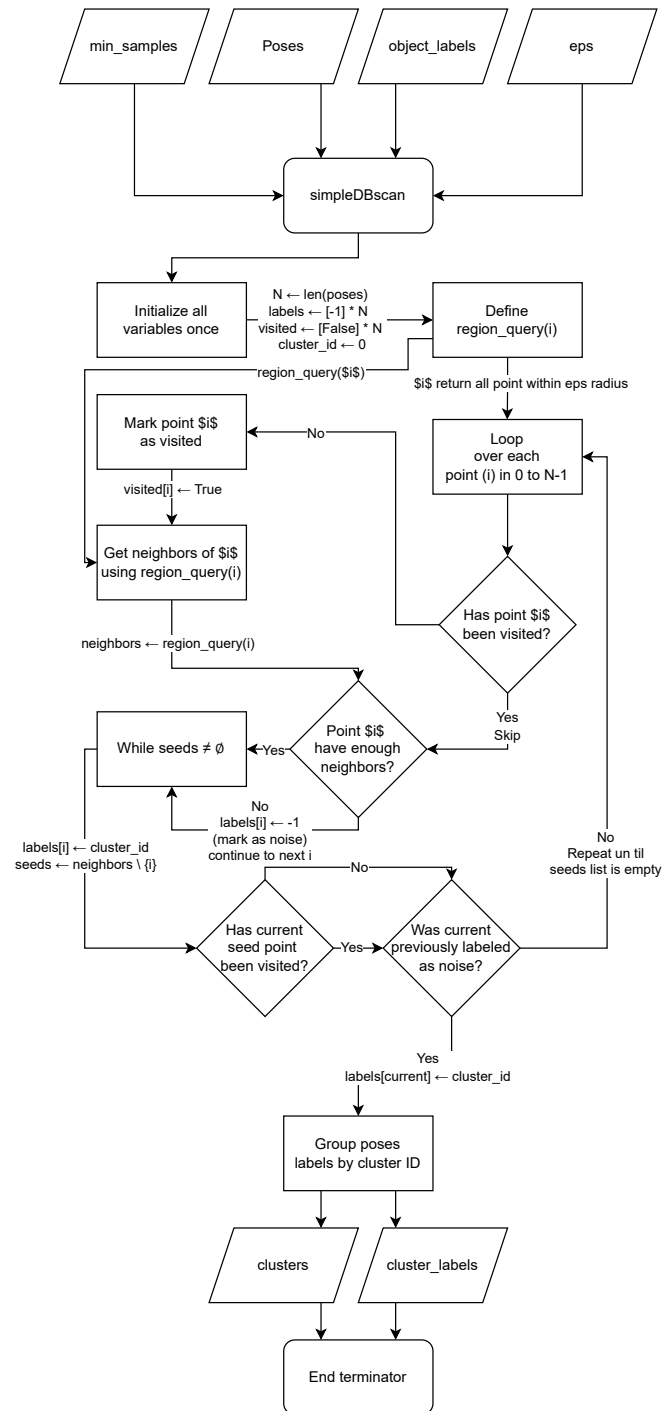


Figure A.2: simpleDBscan.py