
## Instituto Universitário de Lisboa

Department of Information Science and Technology


# Natural Language-Based Human-Robot Control


João Miguel Nunes Bernardo


A Dissertation presented in partial fulfillment of the Requirements for the Degree of

**Master in Computer Engineering**


**Supervisor**

Ricardo Daniel Santos Faro Marques Ribeiro, Assistant Professor, Ph.D.

ISCTE-IUL

**Co-Supervisor**

Sancho Moura Oliveira, Assitant Professor, Ph.D.

ISCTE-IUL


October, 2019

*"Don't be a two pump chump"*

Rodrigo Tavares de Almeida

*"Té Amanhã"*

Kevin Almeida Ramos

# *Abstract*

With the increase in the number of robots arriving at our homes, it is important to find an easy and efficient way to communicate with them. Natural language is the most natural form of communication between humans, so its use by users to communicate with robots, with no knowledge of how they work, is an added value to control them. We present a solution that allows us to interact with a robot in a natural way, so it is easier for human-robot cooperation to happen. This solution includes a method of transforming the information in natural language into controls that a robot can perform. We implement this solution in the specific case of indoor navigation. Finally, we evaluate the performance and interactivity of our approach, by having users, without previous knowledge of robotic control, controlling a robot in a simulated environment.

**Keywords:** Natural language, Human-robot interaction, Robot navigation

# *Resumo*

Com o aumento do número de robôs a chegar às nossas casas, é importante encontrar uma forma fácil e eficiente de comunicar com eles. A língua natural é a forma mais natural de comunicação entre humanos, pelo que a sua utilização em comunicação com robôs, sem conhecimentos de como os mesmos funcionam, para os controlar é uma mais valia. Apresentamos uma metodologia que nos permite interagir com um robô de uma forma natural, desta forma é mais fácil que a interação humano-robô aconteça. Esta metodologia inclui um método para transformar informação de língua natural em controlos que um robô pode realizar. Demonstramos esta metodologia no caso específico da navegação dentro de casas. Em seguida, avaliamos o desempenho e a interactividade da nossa abordagem, com utilizadores, sem conhecimento de controlo robótico, a controlar um robô num ambiente simulado.

**Palavras-chave:** Lingua Natural, Interação Homem-Máquina, Navegação Robótica

# *Acknowledgements*

Quero agradecer ao Instituto de Telecomunicações pelas condições que me proporcinou para a realização desta tese.

Em segundo lugar quero agradecer aos professores Ricardo Ribeiro e Sancho Oliveira por terem sido grandes orientadores e, por terem marcado reuniões semanais graças às quais o trabalho avançou bem mais rápido do que se estas não existissem.

Obrigado ao grupo dos quatro do IT, constituido pelo Bernardo "Expresso" Ribeiro, João "Chicken Little" Pereira, Kevin "Bacon" Ramos, Rodrigo "Roger" Almeida e Robbert "Té Amanhã" DeHaven pelos grandes tempos que passamos e pela galhofa que causamos incluindo o acidente de elásticos de 2019 e as conversas de hábitos de alimentação estranhos.

Obrigado ao grupo dos abraços e carinho, do qual fazem parte a Inês "Nena" Martins, Joana "Pegonha" Vicente, Pedro "Pexor" Dias, Pedro "Ninja" Gonçalves e Tomás "Estalas" Marques pela grande amizade que partilhamos desde o Liceu e pelas discussões às dez da noite.

Obrigado ao João Mendes, Bernardo Pio e Daniel Clemente por me manterem na boa vida, quer seja a consumir derivados de etanol ou a ser altos feeders e flamers no league of legends.

Finalmente quero agradecer a minha familia, principalmente à senhora Maria minha mãe por ser uma super mãe que tendo em conta tudo o que aconteceu conseguiu que não faltasse nada de necessário na vida e à minha irmã Vanessa pelo carinho que partilhamos e por ser a melhor companhia de viagens.

# Contents

**Abstract** v

**Resumo** vii

**Acknowledgements** viii

**List of Figures** xi

**List of Tables** xiii

**Abbreviations** xv

**1 Introduction** 1
- 1.1 Motivation and Research Context . . . 1
- 1.2 Research Questions . . . 3
- 1.3 Objectives . . . 3
- 1.4 Research Method . . . 4
- 1.5 Document Structure . . . 5

**2 Literature Review** 7
- 2.1 Robotic Control Systems . . . 7
- 2.2 Natural Language-Based Robot Interaction . . . 10
  - 2.2.1 Introduction to Natural Language Human-Robot Cooperation 10
  - 2.2.2 Human-Robot Control Solutions . . . 12
  - 2.2.3 Natural Language Processing Frameworks . . . 15
  - 2.2.4 Summary . . . 17

**3 Natural Language-based Robot Interaction** 19
- 3.1 System Pipeline . . . 20
- 3.2 ROS . . . 22
- 3.3 Robot Control . . . 28
- 3.4 Natural Language Processing . . . 35
- 3.5 Summary . . . 43

**4 Evaluation** 45

4.1 Method . . . . . 45
4.2 Results . . . . . 46

**5 Conclusions** **49**

**Appendices** **57**

**A Test Script** **57**

# List of Figures


2.1 NL human-robot cooperation methodologies divided . . . . . 11
2.2 Standford CoreNLP system architecture . . . . . 16
2.3 SpaCy's Pipeline . . . . . 17

3.1 Proposed solution pipeline . . . . . 21
3.2 Topic Architecture . . . . . 23
3.3 Twist Messages Explained as shown in `www.clearpathrobotics.com/blog/2014/09/ros-101-creating-node/` . . . . . 24
3.4 Odometry . . . . . 25
3.5 360 Laser example shown in `http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation` . . . . . 26
3.6 Test Gazebo Map . . . . . 26
3.7 Turtlebot3 SLAM Map . . . . . 26
3.8 MoveBase unstuck procedures from `http://wiki.ros.org/move_base` . . . . . 27
3.9 Final ROS node architecture . . . . . 28
3.10 360º Laser Areas . . . . . 30
3.11 Initial left side measure . . . . . 30
3.12 First door is detected and counted . . . . . 30
3.13 Second Door on the Left is reached . . . . . 31
3.14 End of the Hallway laser detection . . . . . 32
3.15 Natural Language processing pipeline . . . . . 35
3.16 Pure count of words used on the three path test . . . . . 36
3.17 Verb count used on the three path test . . . . . 37
3.18 Word Count for synonyms of entrance . . . . . 38
3.19 Word Count for synonyms of hallway . . . . . 38

4.1 Percentage of Successes Vs Fails at achieving the objective on the three paths . . . . . 47

# List of Tables

2.1 Simulator Comparison . . . . . 9

3.1 Robot status used when playing a queue of commands . . . . . 35

3.2 Sequence of frames for commands with only Action and Direction and corresponding robotic language control . . . . . 40

3.3 Sequence of frames for commands with Action, Direction and Count 41

3.4 Sequence of frames for commands with Action and Destination . . . 41

3.5 Sequence of frames for the Sentence "Go down the hallway" . . . . 41

3.6 Sequence of frames for the command "Go down the hallway and turn right" . . . . . 42

3.7 Sequence of frames for the command "Turn right at the end of the hallway" . . . . . 42

3.8 Sequence of frames for the command "Go down the hallway and enter the fourth room on the left" . . . . . 43

# Abbreviations

**AMCL** Adaptive Monte Carlo Localization

**CCG** Combinatory Categorial Grammar

**CRF** Conditional Random Fields

**DCG** Distributed Correspondence Graph

**HMM** Hidden Markov Model

**LDS** Laser Distance Sensor

**ML** Machine Learning

**NER** Named Entity Recognition

**NL** Natural Language

**NLC** Natural Language human-robot Cooperation

**NLP** Natural Language Processing

**ODE** Open Dynamics Engine

**POS** Part of Speech

**RCL** Robot Control Language

**ROS** Robot Operating System

**SLAM** Simultaneous Localization and Mapping

**SDC** Spatial Description Clauses

# Chapter 1

# Introduction

## 1.1 Motivation and Research Context

Robots are being increasingly used outside the industrial environment as explained in Hagele (2016). With this increase in the number of robots in the domestic environment it becomes increasingly important to find ways to facilitate control by a human, so that he does not need to use complicated interfaces, because the user who needs to control the robot may have some physical impediment that makes it difficult or even prohibits the use of direct control over the robot. A natural way of communicating/controlling robots, allows for a whole new level of robot-human interaction, since humans can interact with robots as if they were doing so with other humans, making speech more fluid and control easier for the human. Since these robots are supposed to be controlled by a human who may not have any experience in the field of robotics as Biggs and MacDonald (2003) show, or programming knowledge, it is imperative to find some new way to control the robots.

An answer to this need for a new form of control is natural language (NL), as shown in Matuszek et al. (2010), which by being already used by humans to give orders, directions and location-related information is also a good way to give orders to robots. But being intuitive and natural to the human being can make it easy to give commands that the robot can not interpret, both due to the size of the command and its complexity. For example, the said words may not be in the vocabulary of the robot if it only has a given number of possible commands.

Having been found the way to control the robots without physical interaction, the next problem to solve is to find a way to transform any voice command given by a human into language that the robot can understand so that it can execute the given command as close to the request as possible. For example, a person says to the robot, "Take this pen to Miguel" and the robot will ask "Where is Miguel?", the person then answers "The kitchen" and the robot, not knowing where that room is, asks the person how to get there. To which the person answers "Exit the room, take the second right turn and you are in the kitchen". With that interaction the person will get the robot to take the pen to the destination it wants it to go, and the robot will understand the location of the room and that the person called "Miguel" is located in that area at the moment.

Natural Language Processing (NLP) is the field of research that searches how to transform natural language into usable information. NLP usually is tied with Machine Learning (ML), which is one of the solutions used to solve the problem which was explained above as shown in Liu and Zhang (2019). When you have a system trained to receive natural language and return the corresponding robotic language the real problem becomes the training that has to be given to the system in order for it to work. Another problem is that the mapping of locations is not always feasible, due to the costs or the time it takes to map a certain location. Natural language can be used as an extra sensor in the robot as implemented in Duvallet (2015)'s solution. This means that the robot has the extract from the users dialog the relevant information for the movement in the environment where it is.

With works like SIRI and Google Assistant already existing and having high detection values as seen in Alagha and Helbing (2019) the true barrier of speech robot control is not converting voice into control but grounding the information from textual input, that is transforming the symbolic objects and situations from the textual input into the objects that exist in the environment the robot is traversing. As such this last topic is the one our work will be focusing in.

This work has its context in the areas of robotics, which is a branch of technology that had its beginning in the last decades of the twentieth century and which was applied to industrial processes in order to increase productivity, both in terms of quality and quantity through the automation of processes, and to increase safety in the workplace for workers.

It is also included in the area of natural language processing which is the language spoken by humans as a basic form of communication.

Associated with the area of natural language processing is the area of machine learning which is the study of recognition of patterns that allows a system to learn through experiences in order to make predictions about data through which it will be possible to transform voice orders into commands that the robot can make.

## 1.2 Research Questions

The main question to be answered by this work is if it is possible for a generic system to be created that allows a robot that is new to an environment to be controlled by a human, giving commands in natural language, as if he was giving commands to another person. This main question can be broken in smaller ones, making it easier to evaluate progress:

- Will the robot be able to, when be given a command in the form of natural language, understand and follow the command even if it is one given with a large amount of orders?
- Can a robot receive and follow multiple commands, including commands that have order/numbers in them, and be able to divide it into smaller tasks it can fulfill and follow all of them?

## 1.3 Objectives

The main objective of the following research is to adapt a robot's software system, in order to allow it to be controlled via natural language by a human who does not have expertise in controlling robots.

This main objective is separated into two parts that are the adaptation of the actual control of the robot, which have to be changed in order to have new actions added that will allow the desired controls to be followed.

The secondary part will be to divide the rich, yet ambiguous, information that natural language contains into smaller bits of information that can be processed

by the robot's controlling software. Finally if the robot either does not understand or cannot fulfill the commands for some reason, he will have to ask for help from the person who is controlling it by asking for a clarification on the order or asking for a new set of orders like Thomason et al. (2015)'s robot does.

## 1.4 Research Method

The following research was done using the Design Science Research Methodology Process Model which is used to design, create and evaluate IT artefacts that can solve known and identified problems. The method is fully described in Peffers et al. (2007).

- Problem Identification and Motivation - The first part of the research is finding the problem which needs to be solved and the motivation to do so, this has been presented in Chapter Chapter 1.1.
- Objectives Definition - After the problem has been defined one must think of a solution for it using knowledge of what is possible. This solution will have multiple objectives, which can be quantitative (e.g., the terms in which a desirable solution would be better than current ones) or qualitative (e.g., a description of how a new artefact is expected to support solutions to problems not hitherto addressed). The definition of objectives has been done in Chapter Chapter 1.3.
- Design and Development - Determine the IT artefact's final functionality and architecture and then create said artefact, which are potentially constructs, models, methods, or instantiations (each defined broadly). This design is seen at the start of Chapter 3 and the system pipeline is the artifact being designed
- Demonstration – Demonstrate the use of the artefact to solve one or more instances of the problem. This could involve its use in experimentation, simulation, case study, proof, or other appropriate activity. The implementation of the solution in a specific case for demonstration is shown in 3 where we implement the proposed solution in a specific case of a indoor navigation robot

- Evaluation – Observe and measure how well the artefact supports a solution to the problem. This activity involves comparing the objectives of a solution to the actual observed results from using the artefact in the demonstration. The evaluation is done in 4

- Communication - Communicate the problem and its importance. Communicate the artefact: its utility and novelty, the rigour of its design, and its effectiveness to researchers and other relevant audiences, such as practicing professionals, when appropriate.

## 1.5 Document Structure

The document is structured in the following way:

- In Chapter 2 we display a research on other works in the area of robotic control using natural language as well as a research on robotic simulation methods and models.

- In Chapter 3 we present our solution for the problem presented in section 1.1 and show some examples on how it could work on various robotic systems. We then show the implementation of our solution in the specific case of an indoor land navigation robot.

- In Chapter 4 we explain how we tested our implementation and show what results we got when testing.

- Chapter 5 summarizes our work, answers the research questions and proposes future work that can deal with the limitations that our implementation has.

# Chapter 2

# Literature Review

In this Chapter, was presented a literature review of the work regarding robotic control systems which was used to help choosing a simulation method and robot model used for the final implementation of our solution.

We also present research on natural language processing including robotic control using natural language commands, textual input parsing into useful information for robot control and natural language processing frameworks which help transform textual input into information which is easier to work into robotic controls

## 2.1 Robotic Control Systems

The first objective of the research was to find methods of controlling robots, both in real and simulated environment. So the initial part of the work was searching for a simulation method. During this research multiple works were found that had open-source simulators available.

Right at the start of the research we found how important Robot Operating System (ROS) is in the area of robotic control. As explained in Quigley et al. (2009), ROS is more than a regular operative system. It is a middle-ware that was designed to provide services like low-level robot control, passing messages between processes and package management. It works in a system of nodes that are individual processes and that are connected by topics, which is the main communication method between topics. These nodes are all connected to a main node

Called ROS Master so this node can setup communication between the individual nodes and the creation of topics. The decentralized architecture works well on robots, leaving the on-board CPU to do small commands and if heavier processing power is needed the robot can use an off-board computer. With ROS being big in the robot community it is a good idea to use it in our main system, since any problem we might have while implementing the robot's control software may have an already existing solution in the ROS forums that have an active community.

SIMBAD, as Hugues and Bredeche (2006) explains, is a 3D robot simulator that works with Java, it uses built-in physics instead of the usual Open Dynamics Engine (ODE). SIMBAD has two additional components, PicoNode which is a graph-based controller that can use neural networks and PicoEvo which is a library which allows genetic and evolutionary programming used in existing works and research projects. Does not use the ROS and is only usable in simulations which is negative if the research ever uses real robots.

VREP, presented in Rohmer et al. (2013), is a simulation framework in which the main simulation loop is a Lua script that calls child scripts that are attached to specific objects in the simulation. Remote API clients can communicate with VREP via socket written in multiple programming languages (C/C++, Python, Java, Matlab), this client-side can be embedded in most hardware including real robots. Implements a ROS node with a plug-in which allows ROS to call V-REP commands via ROS services, or stream data via ROS publishers/subscribers.

As described in Koenig and Howard (2004), Gazebo is a 3D simulation program in which it is easy to create new models and worlds. Gazebo uses ODE physics engine and can use Player's Adaptive Monte Carlo Localization (AMCL) which will be very helpful in robot simulation and, gazebo has the added bonus that it can be worked as a node in ROS environment . It can be used with Rviz, which is a visualization tool of ROS that allows the user to configure and modify the robots as well as display the ROS topics communicated between the nodes such as cameras, and other sensors. it is light and since we can use ROS to control the robots in gazebo simulation it is possible and easy to make a change from simulated robots into real ones.

| Simulator Program | Main Language | ROS Connectivity | Real Robot Capability |
|---|---|---|---|
| SIMBAD | Java | No | Only simulation capable |
| VREP | Lua Script | Connection possible via socket | Client side can be embedded in real robots |
| Gazebo | C++ | Works as a ROS node | Possible with ROS node communication |

TABLE 2.1: Simulator Comparison

Having researched simulation methods, a comparison, which is presented in Table 2.1, was done. The simulation method picked from this comparison ended up being gazebo, as it works as a ROS node.

Working as a ROS node makes it better for communicating with a robot because it allows the minimum middle software since it allows for direct communication with the robot.

Having the simulation medium was chosen, the next step was to pick a robot model that can be simulated and used in the real world if possible. The robot should be lightweight, use wheels as its navigation method and have a way to map the environment it is moving in. The robot should also have its controller exist in the form of ROS nodes, since as explained before this makes it easier to communicate directly with.

Another important part in picking the robot to be simulated is that it has to have a good set of sensors that allow it to know where it is with minimum error because, as shown in Borenstein and Feng (1995), odometry when used on its own is prone to creating errors in the position that the robot believes it has. So it is important for the chosen model to not only have odometry as its orientation method but also have a laser system which will add to the information created by odometry. This laser system will also be important for the robot to understand its surrounding environment.

After a search on robotic models on ROS's robot website[1], we see they have a large amount of available robots for possible use, including biped robots, flying drones and boat based robots. But two robots are shown to be featured robots of ROS and as such we will review them both, those being Turtlebot2 and Turtlebot3.

Turltebot2, is an updated version of Turtlebot, which in itself derives from the turtlesim node that appears on the first tutorials of ROS. Turltebot2 is sold a cheap research usable robot. It has, as sensors, a ASUS Xtion PRO 3D camera

[1]https://robots.ros.org/

and a 110 degrees/second gyro and the hardware includes mounting plates which allows a user to add new hardware such as sensors to it. Turtlebot2 can also be controlled by ROS using the turltebot teleop node and simulated onto Gazebo.

As Guizzo and Ackerman (2017)'s work on TurtleBot3 shows, it is an upgrade on Turtlebot2 using newer ROS versions including . TurtleBot3 is a land vehicle that comes with a 360 Laser Distance Sensor (LDS) which allows it to run Simultaneous Localization And Mapping (SLAM), making navigation easier, the waffle Pi version also has a RaspberryPi Camera. TurtleBot can be simulated using the gazebo and controlled using ROS and RViz to view the simulated turtlebot's sensors at work. The switch from simulated TurtleBot to the real one is also simple, there is only a need to run an additional ROS node that communicates between the existing ROS nodes and the TurtleBot itself.

## 2.2 Natural Language-Based Robot Interaction

### 2.2.1 Introduction to Natural Language Human-Robot Cooperation

Having picked the Simulation Method and robot to test our solution, the next step was to research on how other authors process natural language from speech/text into information that the robot could use to move around in his environment. This section is devoted to that research.

As shown in Liu and Zhang (2019), Natural Language human-robot cooperation can be divided into three separate groups of processing methodologies accordingly to the final objective of the information retrieved. This division can be seen in Figure 2.1 and the three groups are natural language instruction understanding, natural language-based execution plan generation, and knowledge world mapping.

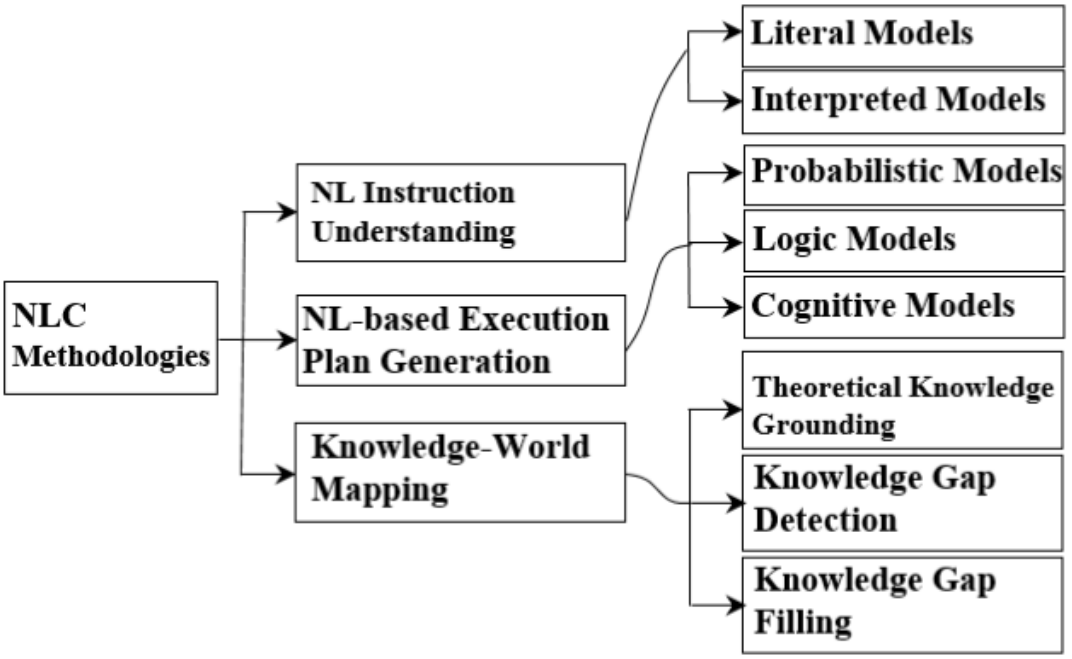



FIGURE 2.1: NL human-robot cooperation methodologies divided Liu and Zhang (2019)

The first group is divided into Literal Models, which include part of speech (POS) tagging, word dependency and sentence syntax. Interpreted Models are also part of the first group, these include Hidden Markov Model (HMM) which models hidden probabilistic relations in linguistic features and Bayesian Network that models probabilistic transitions between task steps. The main difference between the two models is the source of information, because while Literal Models only extract the information from human input Interpreted Models also extract information from the humans surrounding environment.

Natural language-based execution plan generation is divided into Probabilistic Models, an example is a usage of joint probability p(x,y), with which a robot could calculate the probability of the command given being 'move ball' by calculating p('ball','move'). HMM is included once again in this group since they can take into account previous orders and task execution progress in the probability formula. Logic Model is the second type of this group, the idea behind this type is that a natural language task is split in a flow of logic formulas that will fulfill the task itself. The final type is Cognitive Models which uses soft logic which is defined by logic formulas and their weight, that is if a natural language command is partially fulfilled by a robot the command can be successfully executed, a typical model of this type is Markov Logic Network.

The final type of NLC can be divided into theoretical knowledge grounding and knowledge gap filling. Theoretical knowledge grounding methods map learned items which are in the knowledge base of the robot into corresponding objects in real-world scenarios, being defined by visual properties including color and shape/size once captured by the robot's camera these can help it identify objects by identifying objects it can also help the robot understand the division it might be in if it is indoors. Knowledge gap-filling itself divided into gap detection and gap-filling methods, which allow the robot to detect knowledge gaps such as environment constraints imposed by using the robot in a space it hasn't been in before or user gaps, meaning missing information from the controller's side which can be caused by ambiguous or incomplete natural language instructions. Gap-filling methods include using current knowledge in the robot's knowledge base to replace the defective knowledge, using general commonsense, asking for additional human input.

### 2.2.2 Human-Robot Control Solutions

With natural language processing methods divided into understandable groups, we will now show examples of work done by other authors in the area of natural language-based robot control. Including how the information is extracted from natural language and, how the robot systems ground that information into real world situations.

A way for the robot to understand the world around it is to have it create a semantic map based on the information given to it by the controller as explained in Ruiz-Sarmiento et al. (2017). In this work, the robot's knowledge is separated into the Terminological box, which represents the semantic knowledge of the environment (Kitchen is a room and Microwave is an object in the kitchen). The other part of the knowledge is represented in the Spatial box which represents what the robot is seeing with his sensors and cameras, the map in the so-called S-Box allows the robot to pinpoint its exact location on the environment.

Finally by joining the information from the two boxes the author states that the robot makes the semantic map in the form of a *"Multiversal Semantic Map"* ,using Conditional Random Fields (CRF), which contains uncertainty when categorizing viewed objects (0.65 certainty that the object X is a microwave, and if the object X is a microwave the robot has a 0.95 certainty he is in a kitchen). The advantage

given by the author is that using the multiversal map it is faster for the robot to infer information about its surroundings, that it can use multiple sources for the sensors and that the CRF can be retrained to learn new information having in mind the environment the robot will be placed in.

We are shown in Matuszek et al. (2013) a way of making a generic robot control language (RCL) which can then be handled differently by each robot. In this paper, the creators of the RCL parse the NL command received by using a probabilistic type of Combinatory Categorial Grammer (CCG), which models both the syntax and semantics of a sentence and increases the system robustness against the noise found in natural language. This algorithm was trained and tested by using 189 unique sentences generated by non-experts giving directions in two maps supplemented with additional sentences of non-experts giving directions on more complex paths in a third map.

The resulting dataset had 418 NL route instructions including loop and counting, such as 'Go into the ninth door on the right'. This data was segmented into individual movement phrases, re-annotated in RCL and tested on two different maps from the ones used to train, which gave the writer a 66% success rate in short paths and 49% in more complex paths. The way this was tested was by having the robot move from point A to point B on a map following the path given by the controller if the robot reaches the destination via an incorrect path it is considered as a failed trial.

In Paul et al. (2018)'s research work it is shown a way of grounding NL instructions, this is made by first dividing the instruction into smaller constituents, the example given is that the grounding of the command 'pick up the block on the table' is made by associating the constituents 'the block' and 'the table' with the fact that they are objects, 'on' with it being a the region above the object 'table' and that the action picks up involves the robot grabbing the object 'block'.

The possible groundings are divided into Concrete and Abstract, concrete groundings represent real-world objects (box, can, block), regions of space (in front of, behind, to the left), actions (pick up, go to) and ordinal or cardinal numbers (three/third). The abstract groundings represent spatial containers (a group of blocks, row of tables), the spacial context associated with these containers (behind the group of barrels). According to the author taking into account all these objects gives a total number of $17.3 * 10^6$ abstract groundings.

Other authors have show us how robot control is not always one sided. As a human and a robot see and know different aspects of the world around them, the human might give a command that the robot knows it is not possible or is not the most optimal way to get to said goal. As shown in Smith (1991), where the participants have different levels of initiative. One level in which they do not allow the goal to be changed, another in which the participant might suggest an alternative path and a last level in which the participant allows itself to be told which path to be followed.

Another example of mixed initiative robot interaction is shown in Finzi and Orlandini (2005), here the author implemented a system that coordinated a robots' operator interactions with the concurrent actions of rescue rovers. Since the robots can move autonomously the operator can have a higher view on what is happening by being shown by the robots the current state of their exploration. This system allows the user to control the robots if needed be in case a robot is stuck somewhere, but the robots can then keep moving in a self controled way to reach the goal given by the operator.

Another important aspect to have in mind is that natural language can be ambiguous. Duvallet et al. (2016)'s solution deals with this by creating a framework which uses a Distributed Correspondence Graph (DCG) model that extracts the objects and regions that might be in the command given and the relations between them. It then creates a semantic model of the environment using said information and the observations from the robot's sensors, and as the robot moves the created model is updated. The framework includes uncertainty which is updated as the robot moves in the environment, helping the robot overcome the ambiguity problem of natural language.

We have showed how other researchers control their robots by using probabilistic methods and how they ground the information that a human controller can give to a robot into real world objects and movements the robot can understand and produce. Another important part of natural language-based control is dividing the information given by the human into smaller parts which can more easily be parsed by an algorithm that will transform them into robot controls.

Whitney et al. (2017)'s work presents and interesting aproach for dividing this information by using a feedback to collaborative hand-off partially observable markov decision process (FETCH-POMDP), which takes into consideration the

components they call (I, S, A, R, T, O). "I" represents the items on the table which the robot is supposed to interact with. "S" is what the item the controller wants to be grabbed. "A" is the action the robot takes which will give the reward "R" and "T" is the transition function between actions and "O" is the observations on the human language and gestures. So the idea behind this work is if there are two red pens on the table and the controller asks the robot to pick up on of them the ambiguity that exists may be easily taken care of by having the human controller point at which one he wants the robot to pick up. In case the controller doesn't point at the desired pen the robot can always use the action "point()" which consists of the robot pointing at an object and asking if that is the object the human is talking about. This is a good view on how to take care of the ambiguity that natural language can create.

Duvallet (2015)'s research also shows how a command can be divided into the components that make it forming a Spatial Description Clause (SDC). The components of an SDC are a verb, a landmark and the spatial relation to the landmark which can be complemented by a navigation Landmark and the navigation Relation to a said landmark. Dividing the sentences into the parts that make it makes it easier for the information stored in the sentences to be retrieved and as such an important part of the work is knowing what type of division we want to make and what information to get in each division.

Kollar et al. (2010)'s work shows a good inside view into how to gather the information created from the extraction of SDCs, in the work developed they give a good explanation of how natural language works at giving directions. Directions are given in a sequential order, that is, given in the order the receiver of the message is meant to follow and directions will contain information regarding what landmark they refer to. To understand the language used to give directions the authors also created a corpus of their own by asking fifteen subjects to write directions in a set environment the users had never seen before without using the names of the rooms present on the map.

### 2.2.3 Natural Language Processing Frameworks

To help transform textual input into something easier to work with in terms of the information we chose to use a natural language processing framework, which

will aid with tagging raw text and transforming it into a more usable source of information that can be used to retrieve control information from.

StanfordNLP as described in Manning et al. (2014) and Qi et al. (2019) is a framework that provides multiple natural language analysis methods in a pipeline that can be easily extendable by any user. Initially made for internal use was later on extended to allow use to a boarder range of users.

The StanfordNLP pipeline shown in Figure 2.2 is made of a flow of annotators that have behaviors that can be controlled by changing standard Java properties in a Properties object. The flow will tokenize the text, split the sequence of tokens into sentences. Then it labels them with their POS tags, generates the lemma for all tokens and will use a named entity recognition (ner) annotator which recognizes named, (Person, Location) and numerical entities (Number, Time, Duration) and provides a simple framework so the user can incorporate his own Named Entity labels. Finally, the pipeline provides dependency representation based on a probabilistic parser.

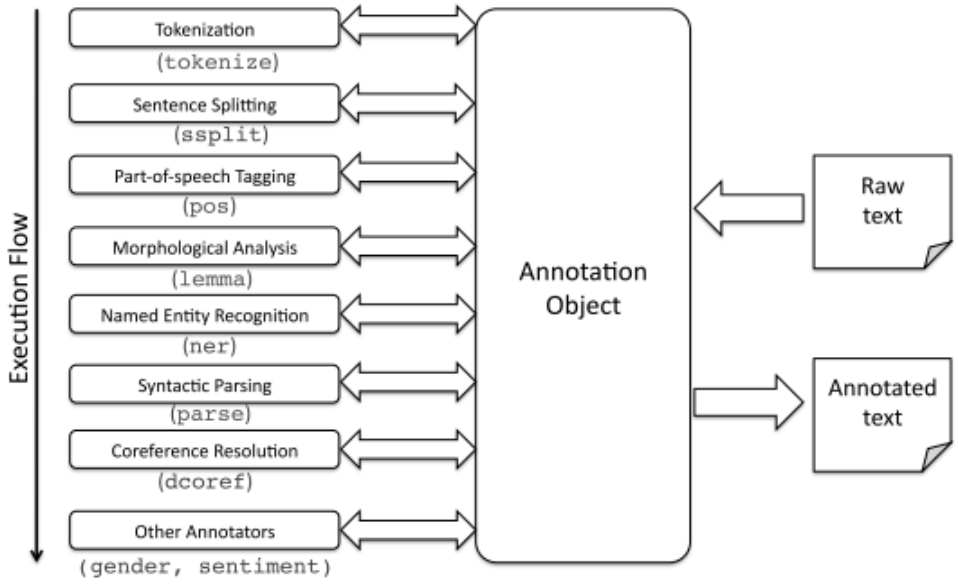



FIGURE 2.2: Standford CoreNLP system architecture
Manning et al. (2014)

The work done in Kurdi (2016) presents us with how the spaCy pipeline works, described as an "industrial-strength natural language processing tool" by its makers. The default pipeline works using a POS tagging algorithm and a NER per

language but it is possible to add more models to the processing pipeline. This makes it light and fast to run being proved in Al Omran and Treude (2017) to be faster than many other NLP libraries. SpaCy is an NLP library made in Python and, as shown in Figure 2.3, as adapted from Spacy's website[2], the first step is tokenizing the text and then tensorizing because spaCy's models are neural network models.

When this step is completed, the next step is POS tagging by a statistical model that can be trained by the user. After that the text is analysed for its grammatical structure and a relation is established between "head" words and words that modify those words, this step is called dependency parsing. The final step in the pipeline is the NER which by default can recognize Persons, Facilities, Countries, amongst other entities. Once all those steps have been done what spaCy outputs is a doc in which is an array of the tokens with their POS tag, NER, and dependencies associated with them.

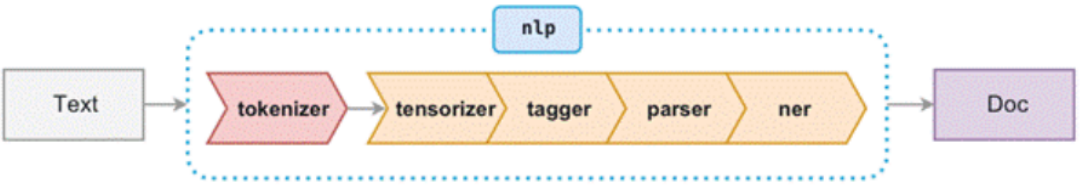



FIGURE 2.3: SpaCy's Pipeline

For our work Spacy was the picked framework since it is lighter in terms of modules and as such faster to be run and since we are going to be working on ROS which uses mainly Python nodes it will be easier to integrate a Python-based framework into it over a Java-based one.

### 2.2.4 Summary

In this Chapter, we have researched methods of controlling robots using natural language-based commands, including on how to simulate an environment and a robot, which is an important first step before stepping into real-world controls. We learned that natural language is an ambiguous source of information and has

[2] https://spacy.io/usage/processing-pipelines

to be parsed and treated in order to allow robot control through it. In most of the related work the authors divided the information from natural language using an object such as SDC or ISARTO as shown in Whitney et al. (2017). Furthermore we discovered that ros is one of the most used robotic languages and that Gazebo is an adequate simulation program since it uses ROS as a form of controlling robot. Turtlebot3 is a great robot to use since it is possible to both simulate it and use it in a physical form, only needing an extra module. While other authors use cameras to aid the robot navigation and control, we are only focusing on the information that the robot can receive from natural language and how it can ground said information.

# Chapter 3

# Natural Language-based Robot Interaction

Having in mind the problems and questions asked in Chapter 1, and the already existing solutions created by other researchers in Chapter 2, we propose a solution that is explained in section 3.1 and implemented in a specific case study in sections 3.2, 3.3 and 3.4.

The implementation can then be divided into three parts. The first being a deeper description of ROS (3.2) in which we explain the software controlling the robot and how the simulation works, including how the data from sensors and controls is created and passed around to be used for controlling the robot.

Secondly in Robot Control (3.3) we show the needed adaptations that were added to the robot controller, and how those adaptations use ROS, in order for it to follow the commands we set out for it to do.

Finally, we have the Natural Language Processing (3.4) part of work, where we explain how the textual input from a human is divided into command blocks that can be passed to the robot software in order for the robot to fulfill the mission it was given.

## 3.1 System Pipeline

The proposed solution is a pipeline where a human writes the natural language commands, which are received by a module whose function is to transform the information in natural language and turn it into controls that the software controlling the robot will receive and send to the robot in a language it can understand and fulfill.

Firstly, the robot's software should have the needed atomic actions added which will allow it to follow the commands and complete the mission it is being used for.

The natural language parsing module is an ROS node whose process will divide the text into its separate components using POS, NER and other tokenization techniques. Once the text is divided into tokens the system will then have a dictionary of verbs that allude to the missions that the robot will have to follow.

Finally, the parser, using the dictionary built for the mission at hand, will divide the already tokenized text into different command blocks, each one being bridged to a robotic action.

Once the natural language has been processed into command_blocks, the natural language parsing module will send the commands, using a ROS topic, to the robotic software system and this system that, in turn, will communicate with the robot itself so it can fulfill the controls given by the human.

So, for example, if the intended mission is indoor navigation the dictionary would have "turn", "left", "right" and "stop" as meaningful words, and an understanding of divisions. The software used as middle-ware between the controller and the robot should have the atomic actions of turning into a said direction, counting doors/turns in a picked direction and memorizing locations.

If the robot's mission is to find people/objects, being in a nautical or aerial environment, important words would be "search", "find", "locate", as well as "heading" and mentions of coordinates and understanding of terrain. In this case the robot's software should have atomic actions which allow it to locate and identify a certain target or to patrol a pre-determined area given by a human, and to communicate back with the person giving it orders.

Finally, if the robot's mission is to grab or fetch items, the dictionary would focus on words such as "bring", "fetch", "grab", "give", "take". In this case the

natural language parsing module would also need to have an understanding of subject and context, for sentences such as "give person X the red pen". So the controller software would need actions that would allow the robot to grab and release items, move them to other locations and, have an understanding of a set of objects and the possible colours

This system allows for a simpler model to be built at the start and if a new action needs to be added, the atomic lower-level action needs to be added to the robot's controller software. Then words categorizing it have to be added to the natural language module, such as action verbs and subjects regarding the said verbs and, finally, a new command block, which bridge the information from the natural language module into the robotic actions, has to be created.

The pipeline for the proposed solution is shown in Figure 3.1.

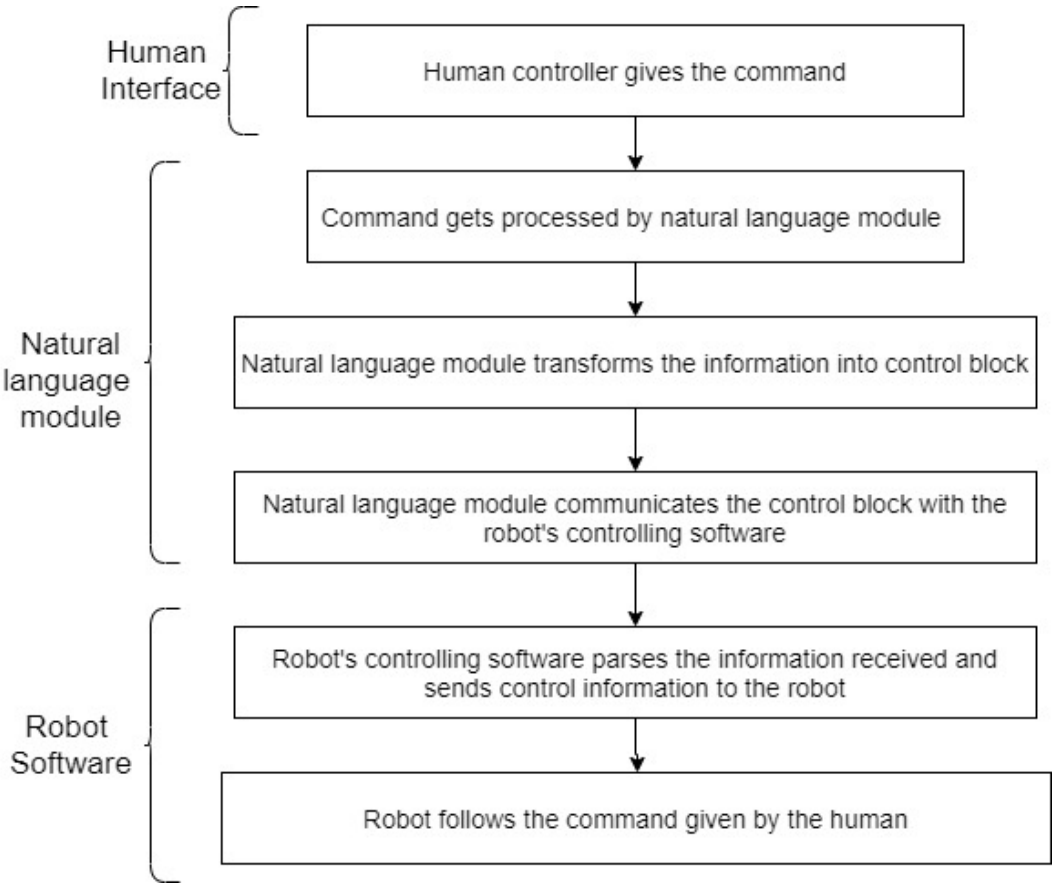



FIGURE 3.1: Proposed solution pipeline

Following the proposed solution we implemented it in the specific case of indoor navigation. To do this we used a gazebo simulation and, for the chosen robotic

model we chose Turtlebot3 model. As we explained in 2, Turtlebot3 has all the necessary traits that will allow us to test our system. The robot's mission is to transverse an indoor environment consisting mainly of corridors and rooms and, is supposed to be able to intake multiple commands at once and follow them.

## 3.2 ROS

As explained in Chapter 2, ROS is a middle-ware which passes information between processes. ROS works in a graph architecture based on nodes that communicate using topics. Nodes are individual scripts that have a specific objective programmed into them and topics are the communication pathway that they use to share information. For example a the node *cmd_vel* exists which is the script responsible for holding information regarding the speed of a robot and using the twist topic a user can change the speed of the robot by telling *cmd_vel* the new speed he wishes the robot to move at.

So in ROS the processing is done in nodes that can receive and post sensor data, control, planning and other messages. ROS processes are nodes connected with each other by topics, nodes can pass messages to other nodes via topics and ask/provide a service to other nodes.

A process called ROS Master knows all the existing active topics, allowing it to setup the topic connections so that nodes can communicate between themselves not needing to communicate through the Master, so messages and service calls are always done between nodes.

Topics are unique named paths that nodes use to send and receive messages. To send a message a node publishes information to said topic, to receive a message from a topic a node has to subscribe to it and when a message is sent to that topic the node will receive it. Messages in topics are anonymous, there is no way to know which nodes are sending or receiving from a topic, the only thing known is what is being sent/received. The types of messages on a topic can be user-defined and include sensor data, state information, motor control commands amongst other types.

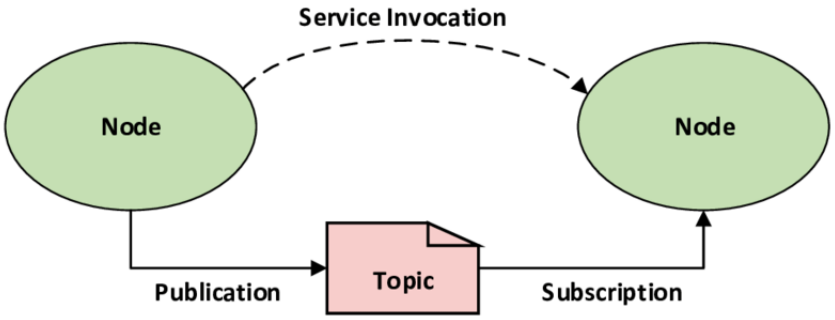



FIGURE 3.2: Topic Architecture
Ploder (2018)

Services are defined, like nodes, by a unique name, and a pair of messages in which one is the request and another the response. Services are actions that a node takes and has a single result, so services are usually used for action that have a defined beginning and end. Services can be seen as a higher-level model than that of topics, since it is synchronous, meaning the client sends a request and waits for an answer, furthermore there is only one service server but there can be many clients. A client can make a persistent connection to a service, which enables higher performance at the cost of less robustness to service provider changes.

Some nodes already come with the Turtlebot3 and ROS installation package, such a the case for gazebo, twist and teleop. This last node is the one we are going to adapt to create singular atomic actions which can be called by textual commands from the robot controller, these modifications done to the node are shown in section 3.3. A Node called NLP was created which parses the command sent by the controller and sends the important command information to the bot via a topic called "NLP_Module". The workings of this node are explained in section 3.4.

The ROS topics being used are the following:

- Clock which is the base time for simulations in ROS that uses because it is easier for the user to change time speed over the simulated environment. All other topics and nodes must run at the same time as /clock topic to avoid synchronization errors.

- Gazebo is the node created by the gazebo program, which was the program chosen to do the simulation and as such is the central node to the simulation, connecting all the other nodes and allowing us to view the robot moving in the environment. Gazebo publishes to the Clock node making the simulation and all nodes connected to it synchronized.

- *Cmd_vel* is the node that holds information about the speed at which the robot is moving, and so it allows the robot to be told to move. This is done by sending a twist message to the *cmd_vel* node and it, in turn, sends it to gazebo so it can simulate the movement $xyz$ on the wheels. Twist messages are broken into a pair of $x$, $y$ and $z$ speeds, one being linear speed and the other being the angular part of speed.

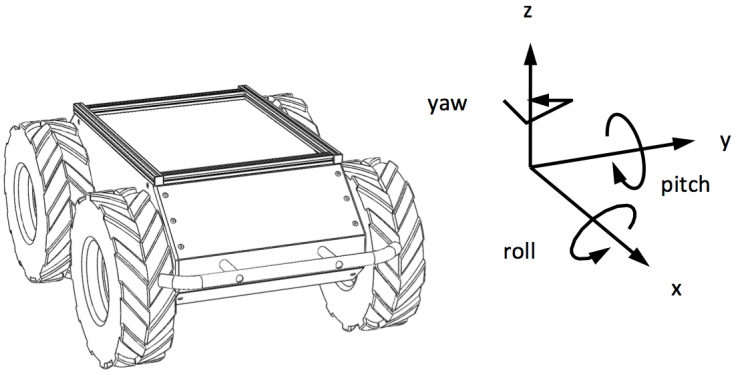



FIGURE 3.3: Twist Messages Explained as shown in www.clearpathrobotics.com/blog/2014/09/ros-101-creating-node/

- Odom is the node that monitors the movement of the robot and uses the data from motion sensors, which count wheel revolution and steering angles, to estimate the position the robot is in relative to it is starting position. An example of an Odometry measure method is shown in Figure 3.4 where the sensor measures the distance moved by the number of times the LED's light is reflected onto it. If used alone in localization it is prone to errors since it is not 100% accurate due to many possible errors occurring in calculating the wheel movement, such as floor irregularity and the extreme probability of wheel slippage as said in Koenig and Howard (2004). The more the robot moves the more errors can happen and these errors accumulate over time.

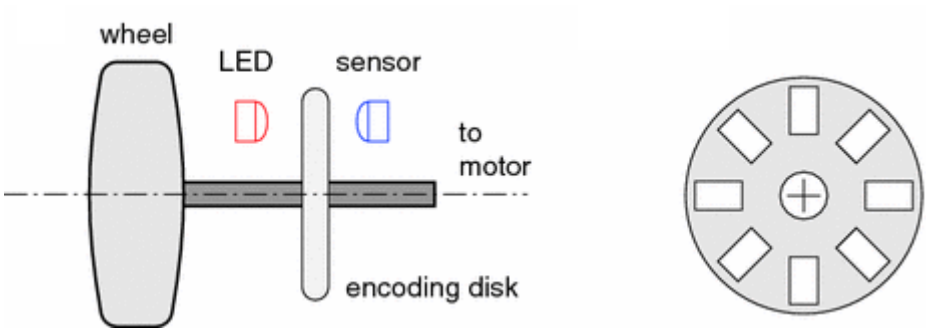



FIGURE 3.4: Odometry
Ben-Ari and Mondada (2018)

- To assist the Odom node on getting a correct localization, Turtlebot3 has a 360 lidar laser scan, which is a laser beam per each degree on the 360º scale, as shown in Figure 3.5, each one measuring up to 3.5 meters in distance and is published into the topic Scan. Using this laser system the turtlebot can employ a karto SLAM algorithm allowing it to map the world around it as it moves, improving on the data collected via odometry which is why usually most robotic systems use both systems if possible in sync. This laser system will be used when checking corners and door/turn counting which makes it possible to tell the robot to "enter the third room on the left" without having the map stored in his local memory.

- To join the laser created map with the information that Odometry gives us, ROS uses the AMCL node which keeps track of where the robot is using a probabilistic localization system for robots moving in 2D and sends that information to the OdomoFrame topic. This is how the robot can get a true sense of where it actually is and how to get back to a known location.

  On the right of Figure 3.7 is shown the SLAM map created using the 360 lidar when the robot took a test drive in labyrinth shown on the left side

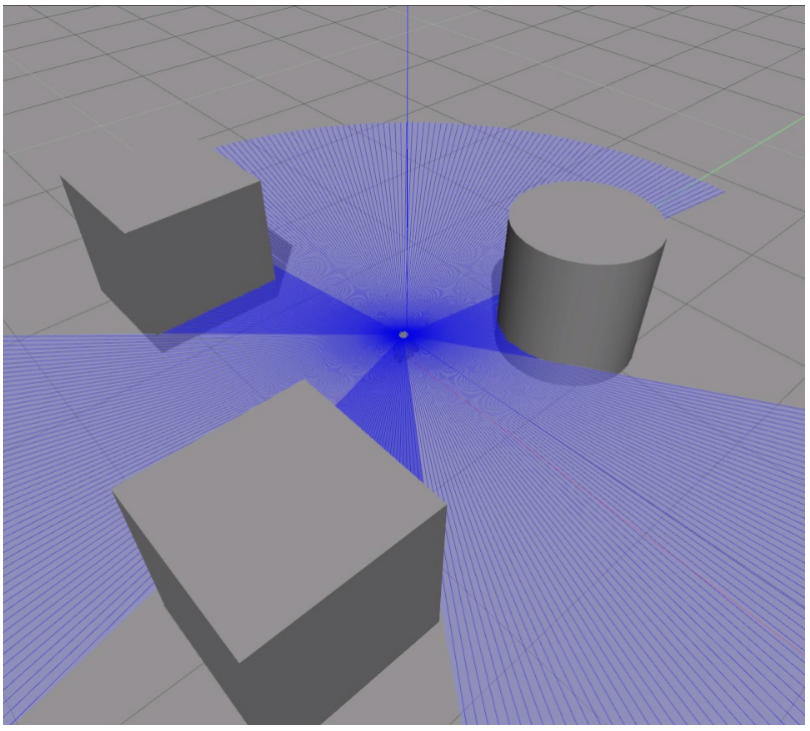


FIGURE 3.5: 360 Laser example shown in http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation

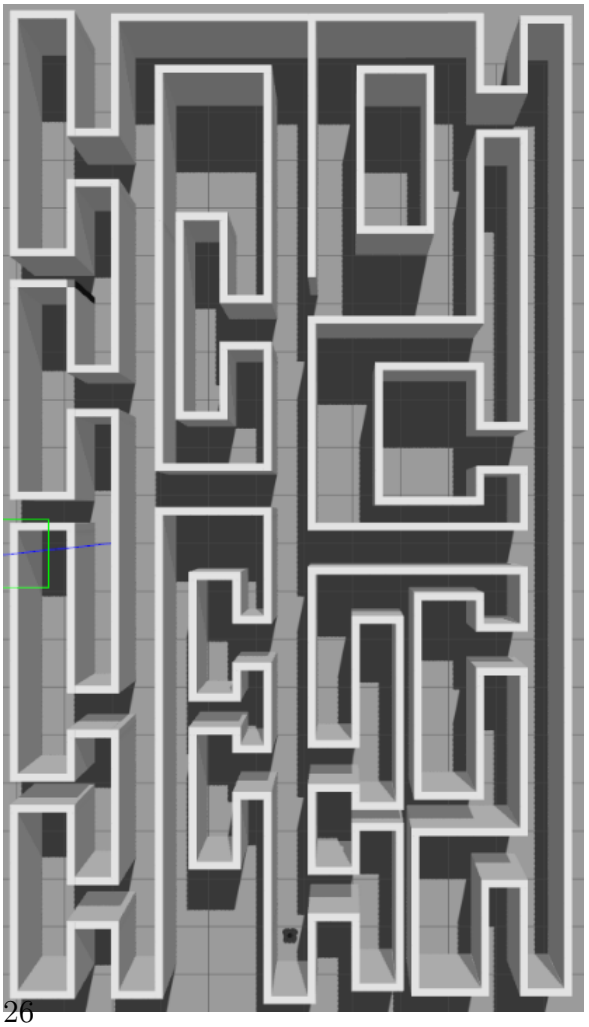


FIGURE 3.6: Test Gazebo Map

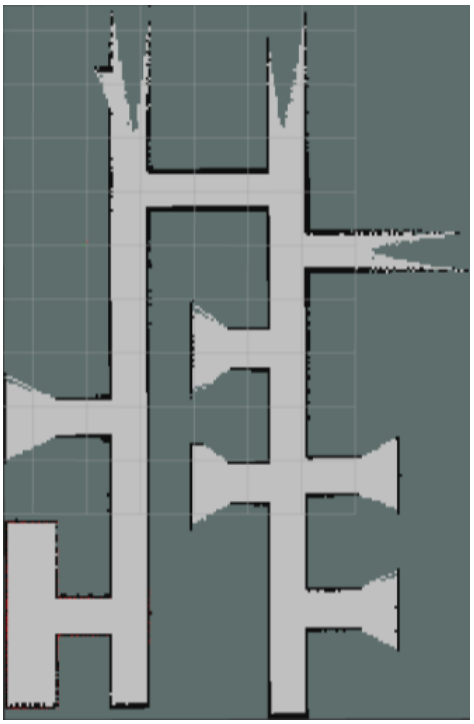


FIGURE 3.7: Turtlebot3 SLAM Map


26

- Tf is the ROS node in charge of keeping track of where all the robot frames (wheels, body, sensors) are at any time in the simulation. So although it is not imperative for the control part or the natural language part it plays an important role in the whole work by allowing us to view the robot model being simulated and moving in the created environment.

- MoveBase node is the major component of the ROS self-navigation stack. An object called goal, which includes the objectives coordinates has to be created and sent to MoveBase node. Once the goal arrives the move-base node will link the global and local planner nodes in order to get the robot to said goal. MoveBase also communicates with AMCL node and any available sensor topic to be able to give the correct controls to get to the destination. The MoveBase also has a series of robot behaviors that will trigger if the robot finds obstacles or finds itself stuck in the environment.

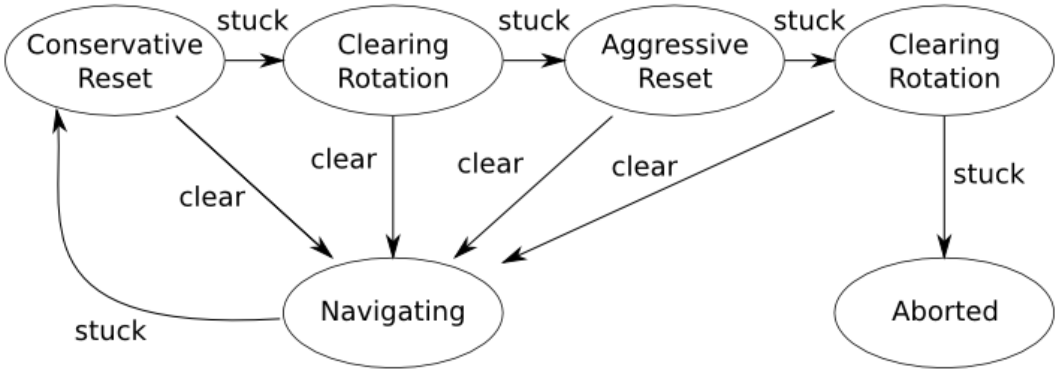



FIGURE 3.8: MoveBase unstuck procedures from http://wiki.ros.org/move_base

The final node architecture including connecting topics is shown in Figure 3.9

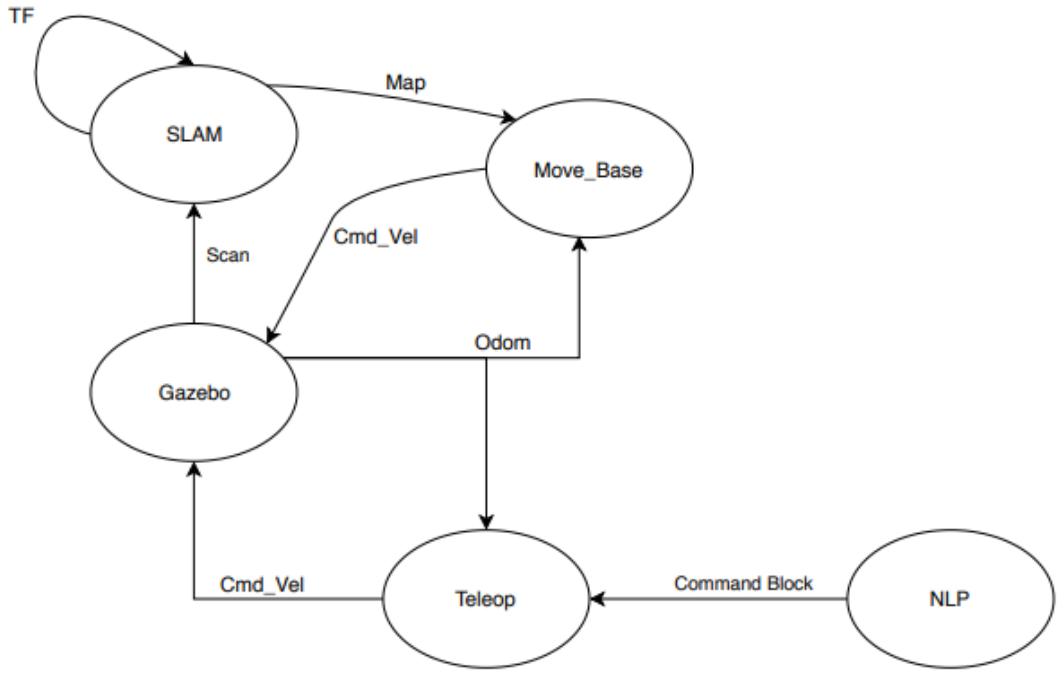



FIGURE 3.9: Final ROS node architecture

## 3.3 Robot Control

In this section is explained what modifications were done to the Turtlebot3 teleop node to allow the full natural language control to be done, which actions were added and what ROS nodes and topics these new actions use as well as how the communication is done with the natural language node, whose workings are explained further ahead in section 3.4

The starting actions that the robot already had available were increasing/decreasing linear speed, which communicates with the *cmd_vel* topic using a twist type message that has in it the new linear speed the robot will have, this will allow the robot to move forward or backward. To allow the robot to rotate left/right the teleop node uses similar mechanisms to those used in linear speed but for angular, decreasing angular speed will turn the robot right and increasing angular speed will turn it left, this is also done by communicating with the *cmd_vel* topic using a twist message with only angular speed this time. Finally, the teleop offers an option to stop any movement the robot is partaking by sending a message to *cmd_vel* in which both linear and angular speeds are at 0.

Since the base actions already existed there was then the need to implement the more advanced controls that would allow a user to move the robot in an indoor environment.

Since the environment the robot will be moving in is an indoor one there will be lots of sharp left and right turns, mainly 90º, and as such there has to be an easier and more precise way for it to turn than just telling it to turn and when the angle seems right telling it to stop. To address that, the first new actions to be added were turning right or left at an angle of 90º, this action used both *cmd_vel* and Odom. First the script calculates what heading the robot has by getting the information from the odom side, then according to the direction the user wants it to turn it either adds or subtracts 90º to that heading and sends twist messages to *cmd_vel* with the max angular speed the Turtlebot3 can do which is *1.82 rads/s* and so the robot starts turning. Once the Odom heading of the robot is at the desired angle the teleop node sends a twist message to *cmd_vel* with the angular velocity at 0 to stop the robot from rotating anymore.

As stated before the robot is going to travel in an indoor environment and we want the human controlling the robot to be able to give him the command to count doors/turns to then enter the division it is turned to. So following the addition of left/right 90º turns as available actions to the robot, the next action would be one that could fulfill the need to count entrances and then turn/enter them. So the next control action done was one that allows the robot to count entrances/turns to either right or left side and then to enter one once it reaches the desired number. To accomplish this the *count_turn* function was created, which first calculates the distance to the desired side by communicating with the laser ROS node via the Scan topic, whose areas are exemplified in Figure 3.10.

Once the initial distance to the desired side (*initial_right* or *initial_left*) is known the robot begins to move forward, and as it moves it keeps calculating the distance to the desired side. Has shown in Figure 3.11 where their order was "turn left on the second door", the robot measures the left side and moves forward. In case the *initial_left*) distance is undetectable or greater than 2, the robot first moves forward and once it reads a smaller distance on the requested size that distance is considered the *initial_left*).

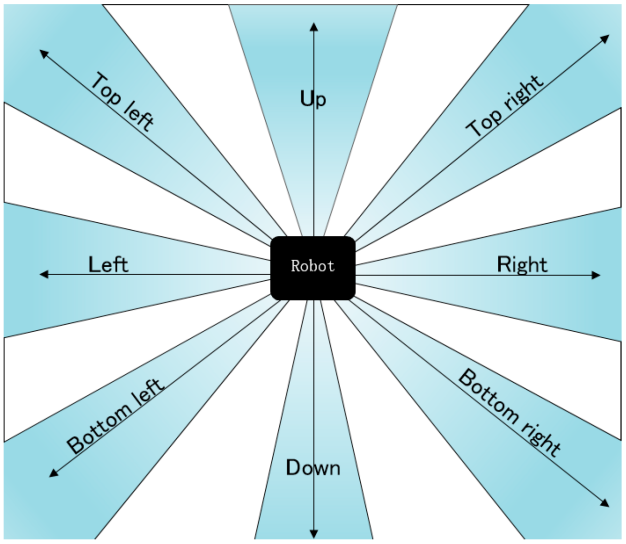



FIGURE 3.10: 360° Laser Areas
Jiang et al. (2018)

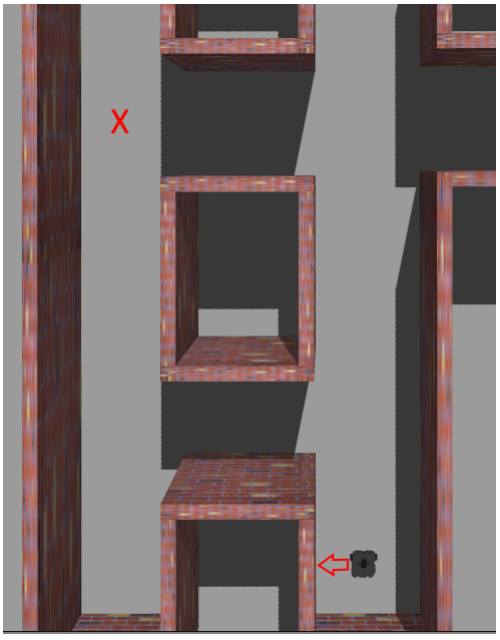



FIGURE 3.11: Initial left side measure

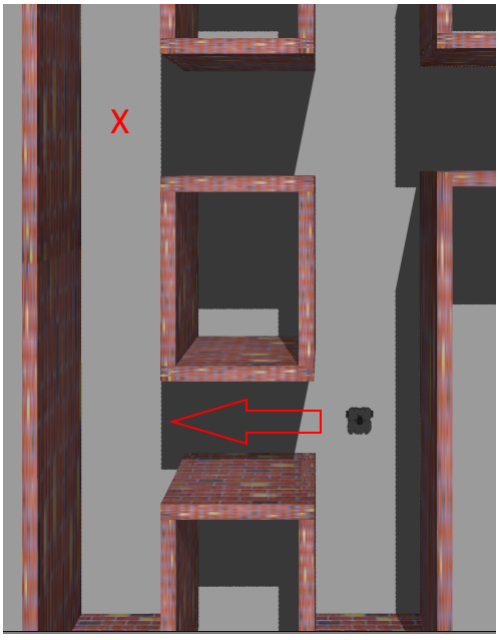



FIGURE 3.12: First door is detected and counted

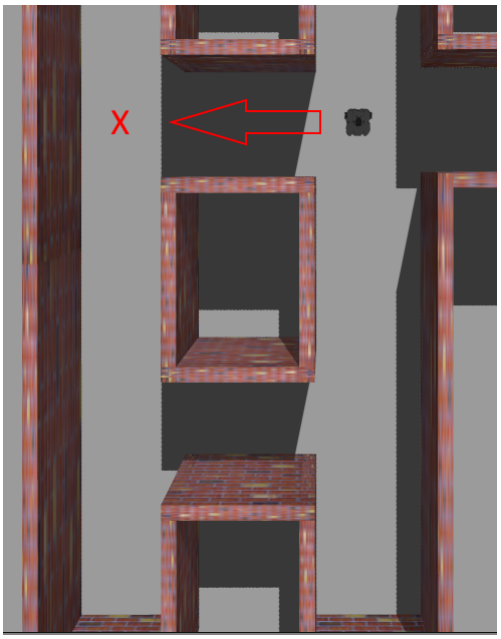



FIGURE 3.13: Second Door on the Left is reached

Once a distance bigger than 1.5 times the starting *initial_right* or left is detected, the distance checker is trigger and adds plus one to the *entrance_counter*, as shown in Figure 3.12. Since the robot will spend some time crossing this entrance there was the need to create a mechanism which would stop the count from going up more than once when it found a door, so whenever the counter is increased a boolean will also be set to false which will stop the number from rising, once the distance is within *initial_right* or left numbers the boolean goes back to true and turn count can continue. When *entrance_counter* equals the desired number of turns wanted by the human the robot is given the order to stop, it then calls the 90º turn action, turning to whichever direction was picked and advances enough distance to be inside the room/hallway it was asked to turn to. This last action allows the robot to recognize it has indeed managed to enter the division it was told to enter after the turn.

Since the robot has a 360 degree laser, the counting algorithm can be strengthen by having in consideration the average laser distance to a specific direction when the robot is moving forward. So if the robot moves in a populated environment a single laser may have his distance shorter then it actually should because it's reflecting of a chair or a person instead of actually reflecting of a wall. By reading multiple sets of lasers at the same time the distance the robot receives is closer to the real distance.

If in the process of moving forward the robot's laser pointing towards the frontal area detects a collision is incoming it stops and send out an error message stating it somehow could not find the turn the user told it to get to, the biggest cause to this error will be asking the robot to get a count of turns that are greater than what the environment permits, such as telling the robot to get to the fourth door on a three-door hallway.

Indoor environments like houses are usually composed by hallways connecting rooms, as such, there is a common command people might give to someone when telling them to move around is to "go down the hallway" or a variant. With this in mind it made sense to make a robot control which allows it to move down a hallway and detect when it got to the end. This was done by using the laser node once more, once the robot receives the command it moves forward along the pathway and, much like in the previous control, once the frontal area of the laser detects a close distance, it stops and understands it has gone down the hallway.

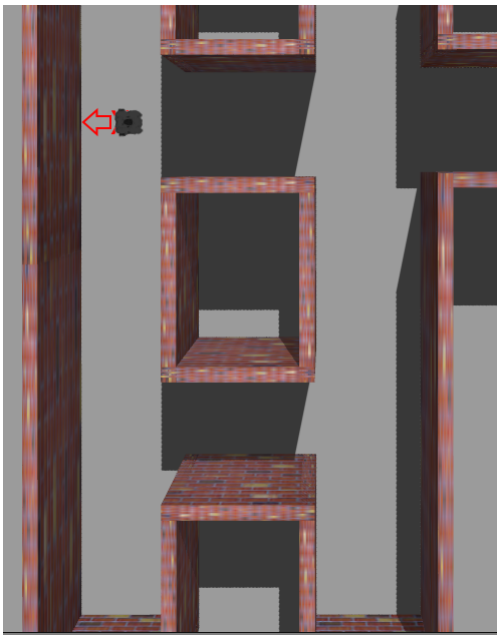


FIGURE 3.14: End of the Hallway laser detection

Following this idea and, in case there are too many turns left or right a person can say "go down the hallway and turn right" instead of saying "turn right on the 10th door", so another mechanism implemented was one which allows the robot to first, like indoor counting, read the lasers received from the Scan topic to see the distance to the desired location. Then as done before go down the hallway and once the frontal laser detects the hallway has ended the robot reads the laser to

the picked side, if the final distance calculated is 1.5 times bigger than the starting hallway distance it means there is an actual turn in that direction, so the robot turns and moves forward into either the room or hallway which it was told to go into. In case the robot reaches the end of the hallway and does not detect a turn towards the desired side it will send an error message saying it could not find the left/right turn at the end of the hallway and requests new commands from the controller.

When giving directions a person might make a mistake and only realise it when the person following those directions is going to the wrong place. A person can say to "Go right" when the desired destination was actually left so it was important to create a control that would allow fixing mistakes. So a control was added which allows the robot to turn back, facing the direction it came from. This was implemented in a similar way to the 90º turn but doubled the angle so the robot would turn 180º.

Like humans remember where they have been and how to get there so can the robot be programmed to do. Following that logic the robot was added the capability to, upon request, save a location's coordinates and the name which describes it in the local knowledge base. This was done by communicating with the Map topic, created by the amcl node, and requesting the robot's current xyz coordinates and saving them tagged with the name the controller picked. The robot's knowledge base of locations can be exported once the simulation is ended and imported when the next simulation begins allowing it to recognize a location by name.

Being told to go to a named division in an indoor environment is normal to a human, and with the robot having the added ability of knowing where a division is and what coordinates it has on the map the following step was to implement and action that once followed would lead the robot to move to said location. This was achieved by first comparing the requested location name with the robot's knowledge base of locations it has saved, and if said location does exist it would then send a goal message to the MoveBase node requesting which way should it move to make it to the destination. Once there the robot will send a message saying arrived at the requested location. If the robot does not know the location's name, it will tell the user that said location is not in its knowledge base and request commands to reach it by saying "I don't know where X place is can you tell me how to get there?". If the robot takes too long to get to the location it will also

tell the user that it was unsuccessfully at reaching it and that it needs further orders to make it there.

Since robots in real case scenarios can slip or have errors in movement, for example a wheel turning more than another or both wheels not stopping at the same time, it was important to implement a passive control that would allow the robot to keep aligned to the environment it is moving it. To allow that we added a method that while the robot is moving first checks the lasers on both sides (*initial_distance_right* and *initial_distance_left*), and while in movement keeps that distance to a margin of 0.1 times the measured distance. If the robot notices the distance is varying it will turn to either side to correct the gap, this way it can stay aligned. This passive control will also activate after turning, where the robot is more prone to errors happening.

The final implementation on the robot control side was a function which connects to the NLP node created for the project, receives the control blocks that were outputted from the textual input and then plays them using the queue function.

The robot can use all the controls stated above via manual control, by using the Turtlebot3 teleop node as it usually runs, which is by pressing the correspondent keys and having the robot do the commands one by one or, the robot can receive a queue of commands and follow them. To allow this a new function was added in which the robot intakes an array of commands, that are written in our generic robotic language, which can be singular commands like "stop" or "turn_right" or dual commands such as ["down_hall_turn", "right"] or ["Count_Right", 3] where both the command and a specification like the number of doors to count or which direction to turn is stated.

To allow the robot to play the commands in a controlled manner, one a time without creating errors in between them, the queue running method has used a set of states which it uses and is showed in Table 3.1. This status also helps a user which has no inside on how the robot works to understand the robot's behaviour easier, because, in the case of an error happening while operating a command, the robot will give output accordingly to what type of error happened.

| Status Name | Explanation |
|---|---|
| Ready | Ready to receive commands |
| Run | In the middle of running a command |
| Turn | In the middle of a turn order |
| Error | Error while doing a command and will request further human input |

TABLE 3.1: Robot status used when playing a queue of commands

## 3.4 Natural Language Processing

This section will cover the work done that covers the information from the point it is received via textual input by a user until the information extracted from it is sent to the robotic part of work done in section 3.3 and this work can be summarised by Figure 3.15

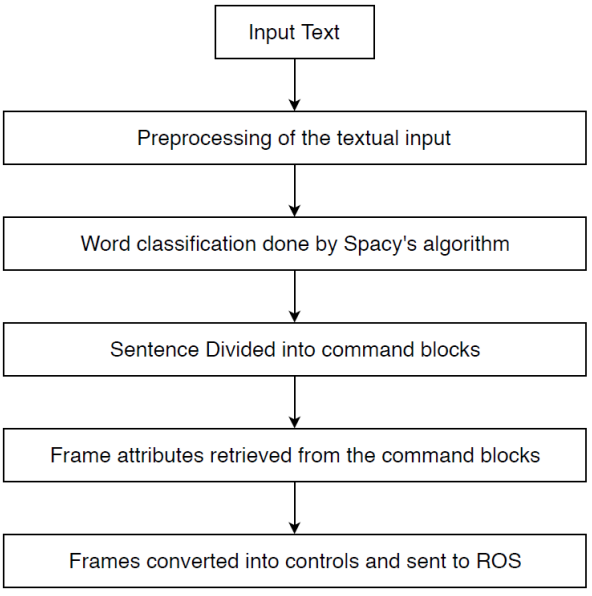



FIGURE 3.15: Natural Language processing pipeline

The first important part of work is to have a dictionary that we can use to help map the textual input into robotic control. With this in mind, we created a small lexicon having in mind the work of Kollar et al. (2010) where a corpus of navigational commands was created from real-world situations by asking people to give directions around the MIT campus.

Our lexicon was created using common direction-giving sentences that can be found in most English books and sites for new learners using verbs and nouns that are associated with indoor house navigation. We then increased the strength of the lexicon by requesting textual input from people in Natural Language and Robotics forums at Reddit. Which we did by creating and sharing a form where we showed various pictures of a robot doing simple commands and asked for textual input which would lead to said actions.

In Figure 3.16 is the raw count of the top 10 single words. Here did not count the words "left" and "right", since this words had to be said by the controller mandatory since they are the only two words defining their directions.

We can see the word "turn" is used by far more than any other word, this can be attributed to the fact that "turn" can both mean the verb "turn" and "entrance", we will divide the word into the two meanings in a later figure.

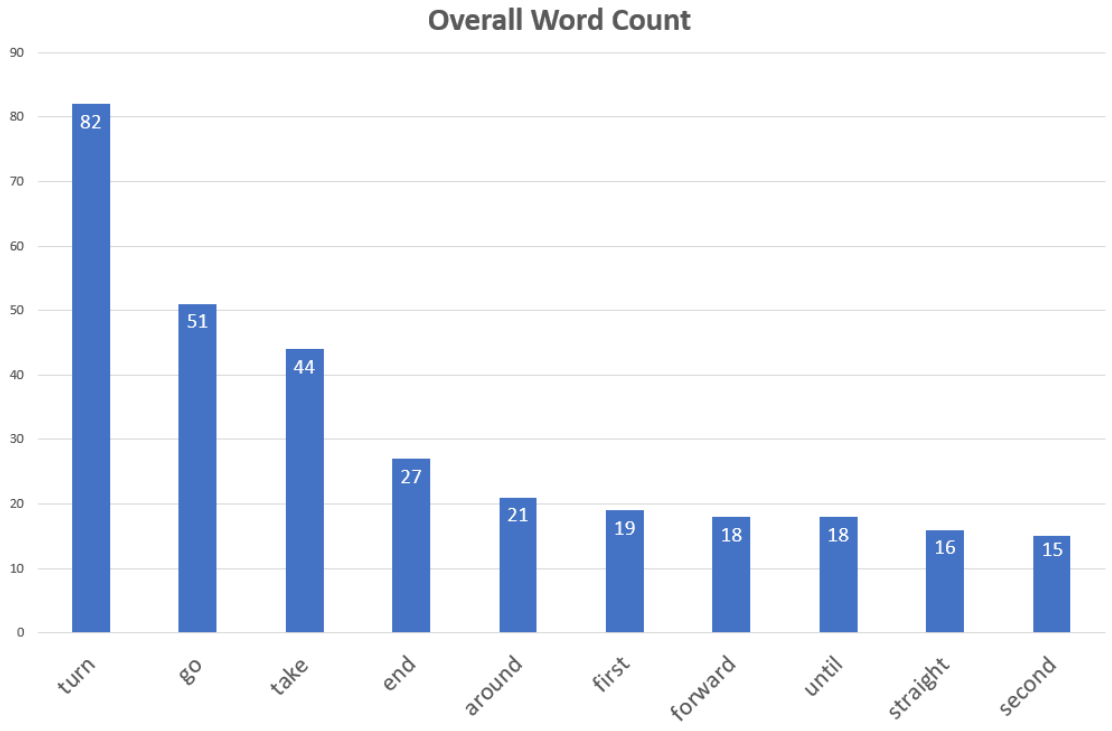



FIGURE 3.16: Pure count of words used on the three path test

Following the distribution by raw word count we now show in Figure 3.17 the distribution for the differently used verbs. We see that "turn" still takes the top spot on the list being the favourite verb used to give navigational orders, followed by "go" and "take" as a close third.

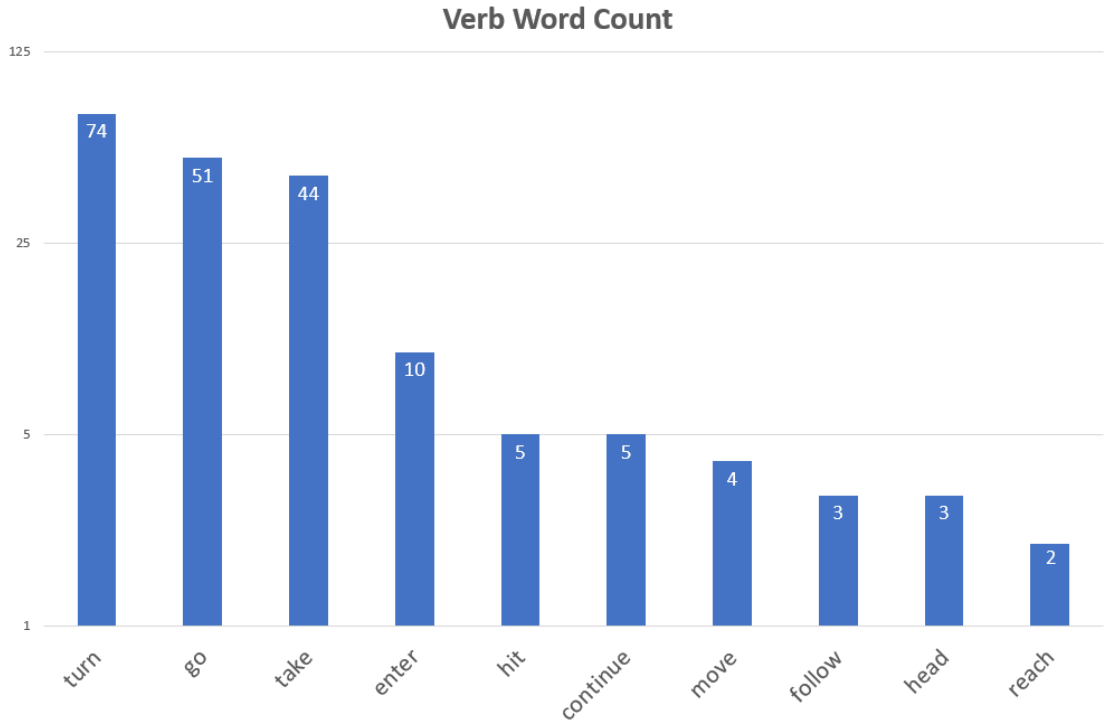



FIGURE 3.17: Verb count used on the three path test

Finally we show in Figure 3.19 what words the users used to name a hallway and in Figure 3.18 we show the different ways an entrance was named. The smaller amount of entrance references can be understood as the controller had the option to tell the robot to enter the first left or right which would get the same effect of going into the room without naming it.

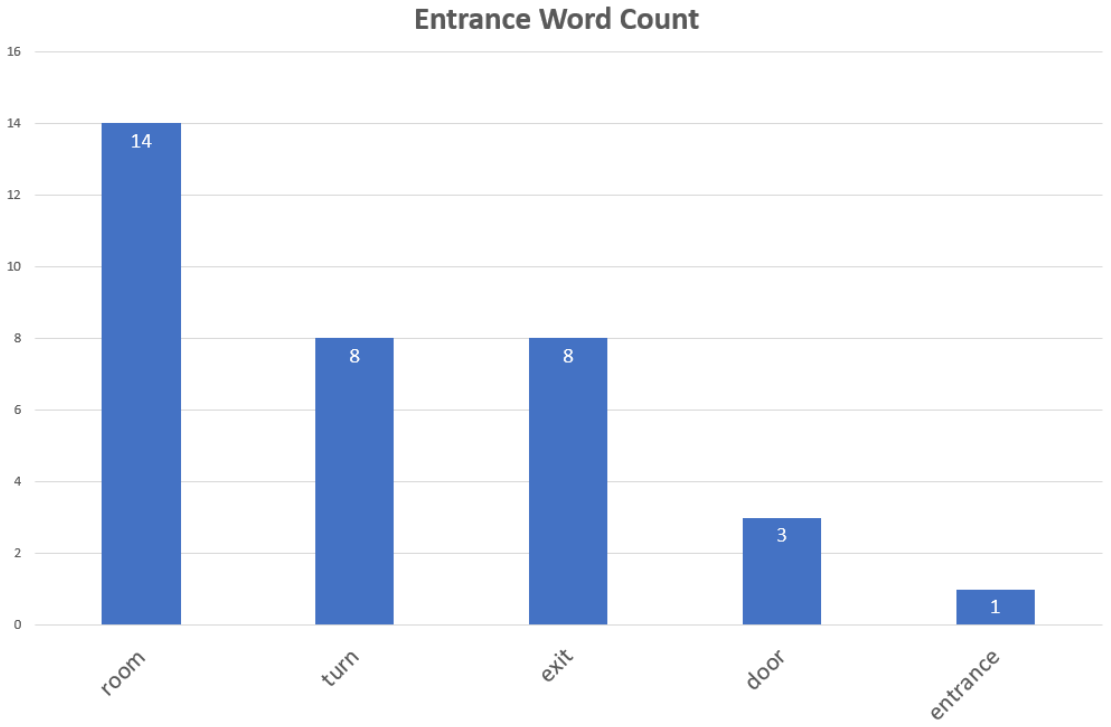



FIGURE 3.18: Word Count for synonyms of entrance

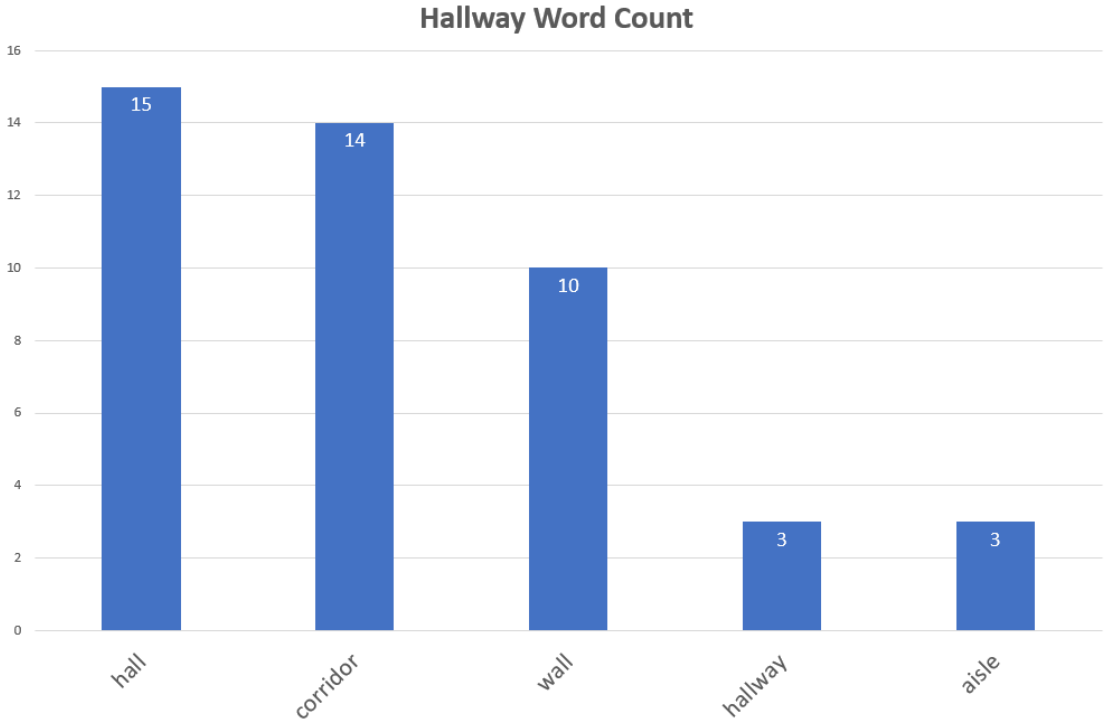



FIGURE 3.19: Word Count for synonyms of hallway

Another important part of converting textual input into robotic control is having a robotic language that is generic enough to be used by multiple robotic systems.

Following that line of thought and as done in Matuszek et al. (2013), we created our own generic robotic language which can be handled by the receiving robot operative system. The creation of a generic language allows our system to be used in various robots, with the only requirement for the implementation being that said robot has wheel based navigation and atomic actions similar to the ones explained earlier on in Chapter 3.3. The next step is processing the textual input by first transforming the text to lowercase and to convert it into Unicode to allow it to be parsed by Spacy's pipeline.

Once the preprocessing phase of work is done we use Spacy, which, as explained in Chapter 2, receives a text as an input and outputs a doc that has the words tagged accordingly to POS tags and dependencies. With this tagged text we can start dividing the sentence using the verbs which are in our dictionary and have a dependency to the root verb of the sentence, to each division of text created by this method we gave the name of command block, and after each command block is processed it will be associated with a control which was developed in section 3.3.

An important part of preprocessing is done after the text is passed through Spacy. This is converting expressions such as "take two rights", which mean turning right on the first possible turn and then repeating said turn when possible, into text which is easier picked up by the algorithm which retrieves the information. With this purpose in mind, we convert any and such expression which have a verb followed by a number and then a direction (left or right) in plural into a repetition of commands which would reproduce such a request.

For example "Take two rights" would be converted into "Take a right and take a right". We also replace any additionality expressions that sometimes replace the verb, which is central to the information extraction process, by an expression which will be easier to parse, similar to the example given before "take a left and another left" is replaced with "take a left and take a left". Finally, we delete any duplicate verbs repeating each other that might be created from the preprocessing.

After dividing the sentences into command blocks the next task was to get the meaning from each of them, to do this we created an object, not unlike Whitney et al. (2017)'s ISARTO, that can be seen as an equivalent of an SDC, which we are

going to call frame and has four attributes. The four attributes are action, direction, count and destination and the different ways these attributes are appointed will say what control will be the output for that frame.

To better get the meaning from a sentence there is the need to have at least two attributes. These can be the action, which is the verb and a direction, that can be left, right, down, forward or back. These two attributes can be complemented by an amount in the form of an ordinal number or a destination (end of the path or a named room). Each command block has one frame associated with it and each frame will usually lead a single control although some frames join together to make a more advanced single robotic control.

If only the action is present it will usually mean a singular action of stopping or moving forward. If there is direction but no number it can mean a 90° turn to the direction side, turning around to face backwards. These textual commands and their robotic language counterparts are shown in Table 3.2

| **Frame** | **Action** | **Direction** | **Robotic Language** |
|---|---|---|---|
| 1 | Move | forward | [increase_linear_speed] |
| 2 | Stop | | [stop] |
| 3 | Turn | right | [turn_right] |
| 4 | Turn | around | [turn_around] |

TABLE 3.2: Sequence of frames for commands with only Action and Direction and corresponding robotic language control

When we get the action, direction and count it can be seen as a command to count the said number of doors or entrances towards the picked side while moving down a path and then once the number has been reached to turn to the direction equivalent to the frame attribute.

Most of the times count will be in the form of an ordinal number but some of the times it can be seen as the word "a" and this is understood to be a request for the robot to enter the first door to the direction side, for example, "Take a right". Accepting this type of command is an important step to increasing accuracy since earlier on we replaced none ordinal numbers where they had the meaning of entering the first X doors in one direction. In Table 3.3 we can see how the attributes are dividing in the frame and how they translate into robotic language.

| Frame | Action | Direction | Count | Robotic Language |
|---|---|---|---|---|
| 1 | Enter | right | third | ["count_right", 3] |
| 2 | Take | left | a | ["count_left", 1] |

TABLE 3.3: Sequence of frames for commands with Action, Direction and Count

In case the attributes we get are action and destination the order resulting from that command block will depend on what action we are faced with. If the action is a *go_to action* the command will be to move the robot to the location with the name of the destination attribute (for example "Go to the kitchen").

Finally, if the action is one which is a synonym of arrival the command will be treated as an order for the robot to save the location it ends its movement in into its knowledge base of locations (such as "You are now in the kitchen"). These orders will also work even if the room is a numbered room like "room 101". The examples regarding the division of these sentences is shown in Table 3.4

| Frame | Action | Destination | Robotic Language |
|---|---|---|---|
| 1 | Go to | kitchen | [go_to_location, "kitchen"] |
| 2 | You have reached | kitchen | [save_location, "kitchen"] |

TABLE 3.4: Sequence of frames for commands with Action and Destination

It can also be necessary to give the command to go to the end of a path and, to allow a said command to be given we also detect commands which have action and destination and in which the direction is a synonym for the end. The importance behind picking up this type of command is also that the more advanced commands which we will be speaking of right after, have a basis on us being able to understand that there is the request to go down a pathway followed by another complementary command. The division of this sentence is showed in Table 3.5

| Frame | Action | Direction | Destination | Robotic Language |
|---|---|---|---|---|
| 1 | Go | the end | the hallway | ["go_down_hall"] |

TABLE 3.5: Sequence of frames for the Sentence "Go down the hallway"

With the simpler singular framed controls taken care of, we now face the more advanced and complicated commands to compute, which are the ones that span across more than one frame.

Those can be the ones with action, direction and destination like in the case of the sentence "go down the hallway and turn right", that has the meaning of moving until the end of the hallway and after that turning to the right and going into the entrance presented in front of us.

This command will be divided into two frames one being the order to go down the hallway and the next one is the order to turn right and to move forward. To detect this type of command every time we find an order to go to the end of the hall the next frame is checked to see if it is a frame referring to a turn action if that is the case only the second frame with produce an order. An example of this can be seen in Table 3.6

| Frame | Action | Direction | Destination | Robotic Language |
|---|---|---|---|---|
| 1 | Go | down | the hallway | |
| 2 | turn | right | | ["down_hall_turn", "right"] |

TABLE 3.6: Sequence of frames for the command "Go down the hallway and turn right"

Another example of complicated commands with action, direction and destination can be seen in the sentence "turn right at the end of the hallway". Here we have a clear order of which direction we want to turn to but, with the added constrain that it has to be at the end of the hallway. Once detected and divided the sentences constituents get placed in the correct attributes and the frame is completed as shown in Table 3.7. The robotic language command is shown as "down_hall_turn" since that is how the robot control system developed in section3.3 expects to receive a control that tells the robot to follow the hallway to the end.

| **Frame** | **Action** | **Direction** | **Destination** | **Robotic Language** |
|---|---|---|---|---|
| 1 | turn | right, end of | hallway | ["down_hall_turn", "right"] |

TABLE 3.7: Sequence of frames for the command "Turn right at the end of the hallway"

The most advanced command is one that has all the attributes, such is the case of the sentence "Go down the hallway and enter the fourth room on the left". This case once more is dealt with by dividing the command into two frames, one being

the order to go down the hallway and the second is the order to count four rooms on the left side and then enter the fourth one to that side.

| **Frame** | **Action** | **Direction** | **Destination** | **Count** | **Robotic Language** |
|---|---|---|---|---|---|
| 1 | Go | down | the hallway | | |
| 2 | enter | left | | fourth | ["count_left", 4] |

TABLE 3.8: Sequence of frames for the command "Go down the hallway and enter the fourth room on the left"

Finally to add to the interaction with the human controller, if for some reason no control is detected in the sentence given by the user the input module will request for the command to be given in a different way if possible allowing the algorithm to once more try to extract meaning from the new command.

Using our frame system most sentences can be broken down into its basic constituents and from that information can be extracted, as such the interaction between a human and our robot will feel more like talking to a human than talking with a robot.

## 3.5 Summary

In this Chapter we show an example implementation of our proposed presented at the start of the Chapter. We do this by implementing the pipeline using a gazebo/ROS based simulation with a Turtlebot3 as the chosen model to run.

We added the required atomic actions to the robot's controlling software using ROS sensors to allow the robot to do the mission it was intended to do.

Finally, we explain the process behind the transformation of natural language input into a generic robot control language, that we created, which includes the pre-processing done to the input, the division of the text into command blocks using a dictionary created from real human input and what information we extract from this blocks to create the controls that are sent to the robot.

With our implementation a person could tell a robot "Bring this object to the kitchen", since this is the robot's first time in this environment and does not know where the kitchen was would ask "Can you tell me how to get to the kitchen?"

and the person would say "Go forward and take a right, then at the end of the hallway go left and you are at the kitchen". The robot can break apart the persons speech into the different commands using our frame system and follow the orders and once at the kitchen it can save the coordinates on his map so next time it knows where it is.

# Chapter 4

# Evaluation

In this chapter we show how we tested our implementation of the proposed solution shown in chapter 3 and what results we achieved when testing. Finally we show word distribution graphs on how different was the input from people when giving the same set of orders.

## 4.1 Method

The objective of the work was for humans with zero experience at controlling a robot to do so by using commands in natural language, as if they were talking to another person. So the important part to test in our implementation was how well can the robot receive multiple commands at once, given in a natural way, and follow said commands without failing.

To answer this question, we made a new simulated map and tested with specific paths, which would lead the robot through all of the map using the available controls. The robot will be given controls by people who had never seen the map, did not know anything of our natural language processing method and were told to give the robot help getting to the desired place as if a person was asking them how to get there. The test was done with 20 different people from ages 18 to 27, with courses not in the area of computer engineering or robotics, and the questions asked to them are shown in Appendix A.

The person controlling the robot would first have to get him to a room which is located on the second left door the robot would encounter when going forward.

The second path would need the robot to turn around and move forward taking the second right and then the first left. Finally the last path needed the robot to turn around and take the first right, then the first left and first left again and, finally, move down the hallway to the starting position.

We wrote down the input and only counted as a correct if the sentence the user gave out managed to get the robot to the requested room following a said path. This test would should how robust our natural language processing system is when faced with input from different people.

## 4.2 Results

Following the tests explained in section 4.1 the results show that eighteen out of twenty users managed to get the robot to to "Room B" correctly following the requested path. It is an expected result since the first command is a smaller one and, as such, should not vary much between controllers.

On the second path the number of users managing to get the robot to "room C" in one command went down by one, meaning seventeen people got the robot to the correct room in one textual command, this reduction of corrects can be attributed to to increasing amount of turns the human has to request in a single command, with people giving orders in way that are out of the range of our system to pick up.

Finally, for the third and final path the results are sixteen successes at reaching the objective in a single command. With the increased amount of turns, and size of the possible commands, that can be given in different ways. It is no surprise this path gets the worst result out of the three. The results of all the tests are shown in figure 4.1

An example of a sentence that failed on the first path was ,"go forward, ignore a left and then enter the next room on the left". This command failed because the natural language processing node does not include the word "ignore" nor does the robot have the capability to deal with it, a possible fix for this would be to preprocess sentences containing the word "ignore" by adding one to the amount of doors the robot was told to ignore. So the command "ignore a left and then

enter the next room on the left" would become something like "enter the room on the second left"

Another example of a failed command for the third path was ,"Turn around and take a right, then take a left and go forward, before you enter the room turn left and go down the hallway". This command did not succeed because the robot does not have commands to deal with "before you enter the room". For this to work would require context of localization and could be fixed with a change our frames of natural language by adding context to the destination attribute.

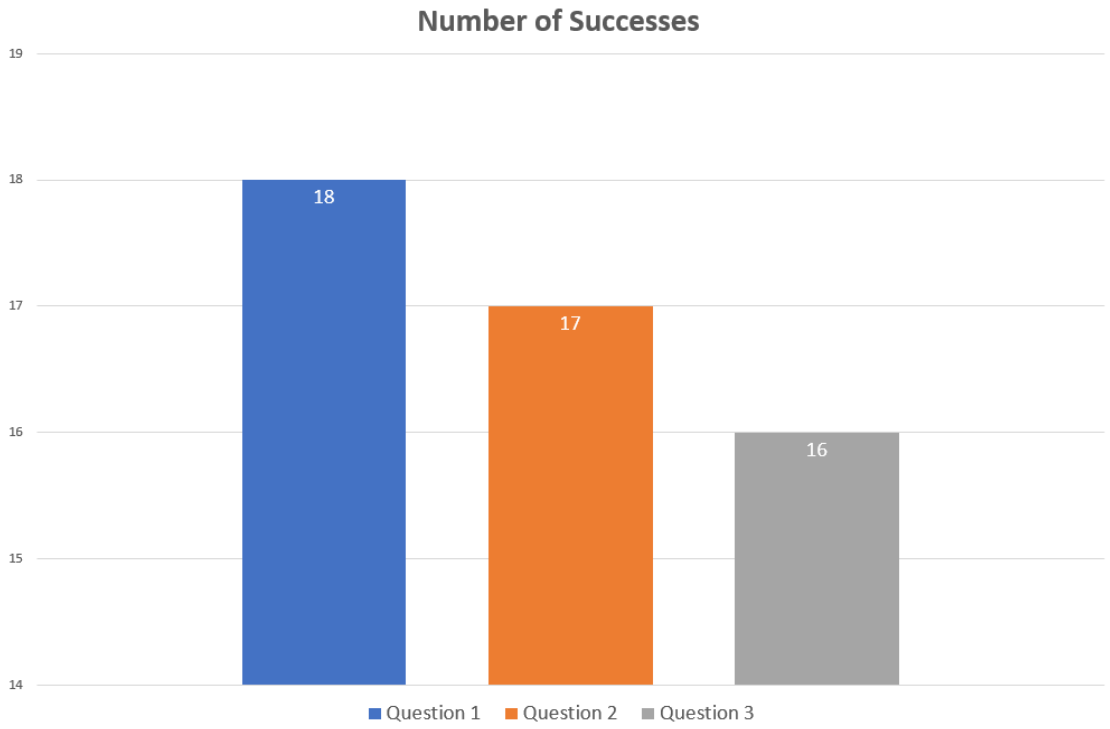



FIGURE 4.1: Percentage of Successes Vs Fails at achieving the objective on the three paths

# Chapter 5

# Conclusions

In this work we show research on how other authors control robots using natural language as a control medium, and how they parsed said natural language when faced with the ambiguity it presents. We then researched robot control methods and found ROS as the most used robot language and that Gazebo is a good simulation medium when using ROS.

We then proposed a system that receives human input in the form of written natural language, parses said language retrieving any useful information for robot control and then sends it to a adapted robot control software.

With our proposed solution shown we then implemented it in the specific case of indoor robot simulation using a simulated Turtlebot3 as the selected robot and Spacy as the algorithm chosen for parsing the information from natural language.

Finally we tested our implemented system by using input from people who have never had contact with our system. In these tests we sought to find out how robust our natural language parsing system was when faced with input from various people with different backgrounds.

From the experiment show in Chapter 4, we can see that it is possible for a person to interact with a robot using natural language as if they were talking to a human and that using the information from this interaction it is possible for human-robot cooperation to happen.

During testing we also noticed the usage of the words "ignore" and of regions of space like "before", which we had not build our system to deal with. We also

noticed that the ideal way for more data gathering to be done would be to create a testing simulation which users could try online and that would get more data on how people give commands allowing us to strengthen even further our dictionary and to add new robotic controls.

With our usage of frames to divide the information contained in natural language, it is possible for an objective to be divided into the smaller commands contained in it, which will allow the robot to follow said commands no matter how long these might be, as long as the verbs defining them are in the system. And from the results shown in Chapter 4 we can also conclude that a robot can be given a larger order containing ordinal or cardinal numbers and that he will follow that order correctly.

Since the system is connected to the robot using the ROS node system we can also conclude that the methodology behind our solution will also work on other robots that have ROS as their main software, as long as the atomic actions defining our objective for the robot are in the controller software and the robot's navigation method is similar to ours. This increases the number of robots we can communicate with by a large amount.

For future work we would like to change the script behind the natural language node, by using a machine learning algorithm that could be taught how to understand the way humans formulate commands and the planning behind those commands. Doing so would improve the rate at which the robot understands all sorts of commands for any situation. We would also like to implement our proposed solution on real robots, and robots that navigate in a different way than our robot.

# Bibliography

Al Omran, F. N. A. and Treude, C. (2017). Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 187–197. IEEE Press.

Alagha, E. C. and Helbing, R. R. (2019). Evaluating the quality of voice assistants' responses to consumer health questions about vaccines: an exploratory comparison of alexa, google assistant and siri. *BMJ health & care informatics*, 26(1).

Ben-Ari, M. and Mondada, F. (2018). *Elements of robotics*. Springer International Publishing.

Biggs, G. and MacDonald, B. (2003). A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, pages 1–3.

Borenstein, J. and Feng, L. (1995). Correction of systematic odometry errors in mobile robots. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 3, pages 569–574. IEEE.

Duvallet, F. (2015). Natural language direction following for robots in unstructured unknown environments. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST.

Duvallet, F., Walter, M. R., Howard, T., Hemachandra, S., Oh, J., Teller, S., Roy, N., and Stentz, A. (2016). Inferring maps and behaviors from natural language instructions. In *Experimental Robotics*, pages 373–388. Springer.

Finzi, A. and Orlandini, A. (2005). Human-robot interaction through mixed-initiative planning for rescue and search rovers. In *Congress of the Italian Association for Artificial Intelligence*, pages 483–494. Springer.

Guizzo, E. and Ackerman, E. (2017). The turtlebot3 teacher [resources_hands on]. *IEEE Spectrum*, 54(8):19–20.

Hagele, M. (2016). Robots conquer the world [turning point]. *IEEE Robotics & Automation Magazine*, 23(1):120–118.

Hugues, L. and Bredeche, N. (2006). Simbad: an autonomous robot simulation package for education and research. In *International Conference on Simulation of Adaptive Behavior*, pages 831–842. Springer.

Jiang, L., Zhao, P., Dong, W., Li, J., Ai, M., Wu, X., and Hu, Q. (2018). An eight-direction scanning detection algorithm for the mapping robot pathfinding in unknown indoor environment. *Sensors*, 18(12):4254.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

Kollar, T., Tellex, S., Roy, D., and Roy, N. (2010). Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 259–266. IEEE Press.

Kurdi, M. Z. (2016). *Natural language processing and computational linguistics: speech, morphology and syntax*, volume 1. John Wiley & Sons.

Liu, R. and Zhang, X. (2019). A review of methodologies for natural-language-facilitated human–robot cooperation. *International Journal of Advanced Robotic Systems*, 16(3):1729881419851402.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Matuszek, C., Fox, D., and Koscher, K. (2010). Following directions using statistical machine translation. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 251–258. IEEE.

Matuszek, C., Herbst, E., Zettlemoyer, L., and Fox, D. (2013). Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer.

Paul, R., Arkin, J., Aksaray, D., Roy, N., and Howard, T. M. (2018). Efficient grounding of abstract spatial concepts for natural language interaction with robot platforms. *The International Journal of Robotics Research*, 37(10):1269–1299.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77.

Ploder, O. (2018). *SecUAV - A Unified Testbed for the Evaluation of Secure State Estimators*. PhD thesis.

Qi, P., Dozat, T., Zhang, Y., and Manning, C. D. (2019). Universal dependency parsing from scratch. *arXiv preprint arXiv:1901.10457*.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Rohmer, E., Singh, S. P., and Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE.

Ruiz-Sarmiento, J.-R., Galindo, C., and Gonzalez-Jimenez, J. (2017). Building multiversal semantic maps for mobile robot operation. *Knowledge-Based Systems*, 119:257–272.

Smith, R. W. (1991). *A computational model of expectation-driven mixed-initiative dialog processing*. PhD thesis, Duke University.

Thomason, J., Zhang, S., Mooney, R. J., and Stone, P. (2015). Learning to interpret natural language commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Whitney, D., Rosen, E., MacGlashan, J., Wong, L. L., and Tellex, S. (2017). Reducing errors in object-fetching interactions through social feedback. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1006–1013. IEEE.

# Appendices

# Appendix A

# Test Script

Instituto Universitário de Lisboa

Departmento de Ciências e Tecnologias da Informação

Masters in Computer Engineering

# Human-Computer Interaction Test

João Miguel Nunes Bernardo

**Supervisor**

PhD. Professor Ricardo Daniel Santos Faro Marques Ribeiro

ISCTE-IUL

**Co-Supervisor**

PhD. Professor Sancho Moura Oliveira

ISCTE-IUL

October 2019

# Introduction

## What is the master thesis about?

The work being tested is robot control via natural language. The simulated robot is built to receive textual input in English and move around an indoor environment, and as such can understand vocabulary related to navigation in said domain. The environment the robot is being tested on is a three room two hallway ‘house’. The robot starts in an entrance and it is supposed to be moved by the user around the environment.

## What is being tested?

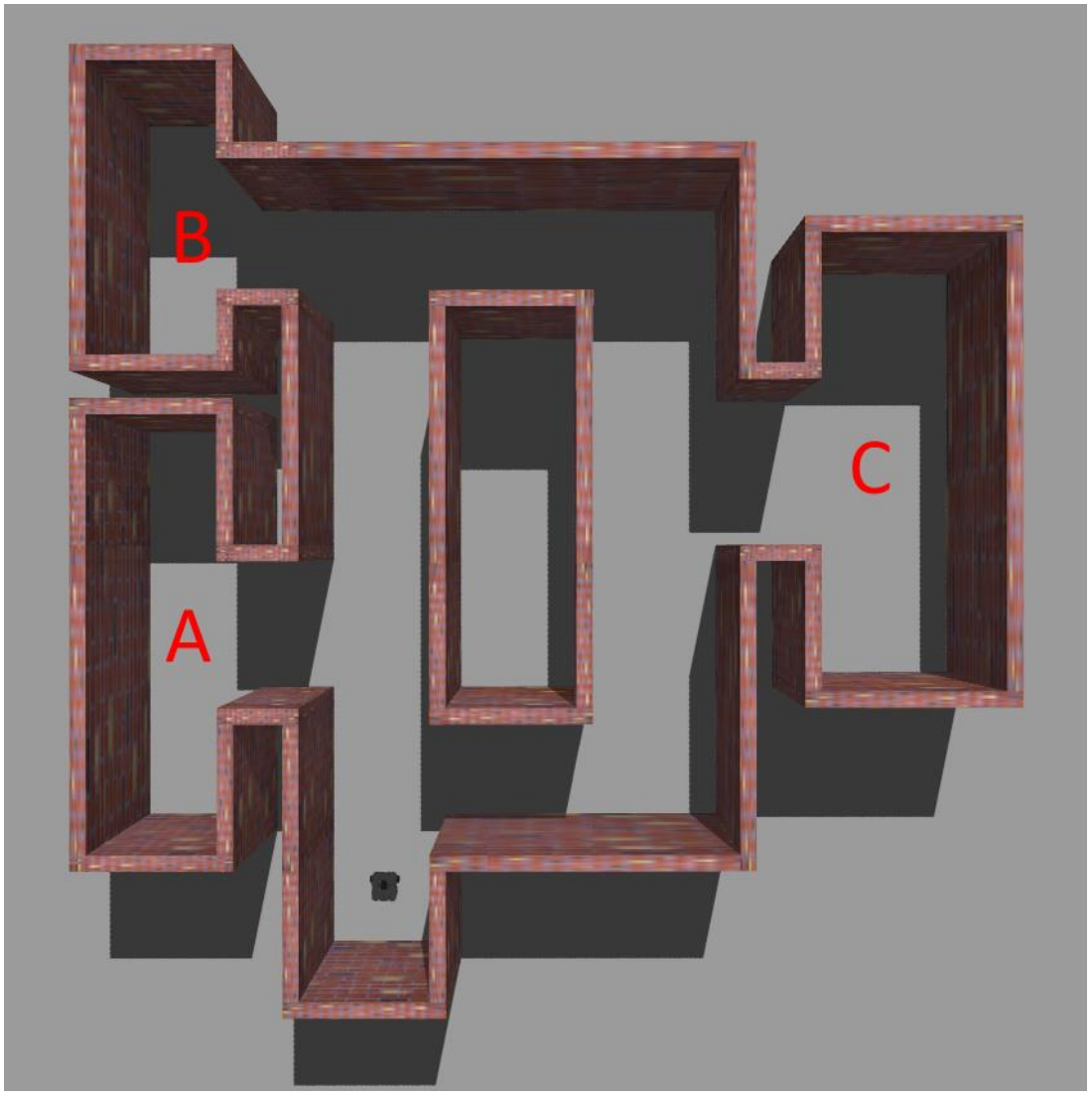
The part of the work being tested is the capabilities of the robot to understand the textual input and transform it into working commands. Since different people give navigation orders in different ways it is important to know how robust the robot is to these changes in controller.

## What do you need to do as a tester?

Once you are done with reading these instructions and agree to do the tests, the simulation will start, and you will have to tell the robot how to get to the objectives as if you were telling another person how to get there. Once the objectives have been fulfilled, the test is complete.

By doing the tests you have knowledge that the data being collected is fully anonymous and it will only be used for academic/scientific uses.

# Navigation Map



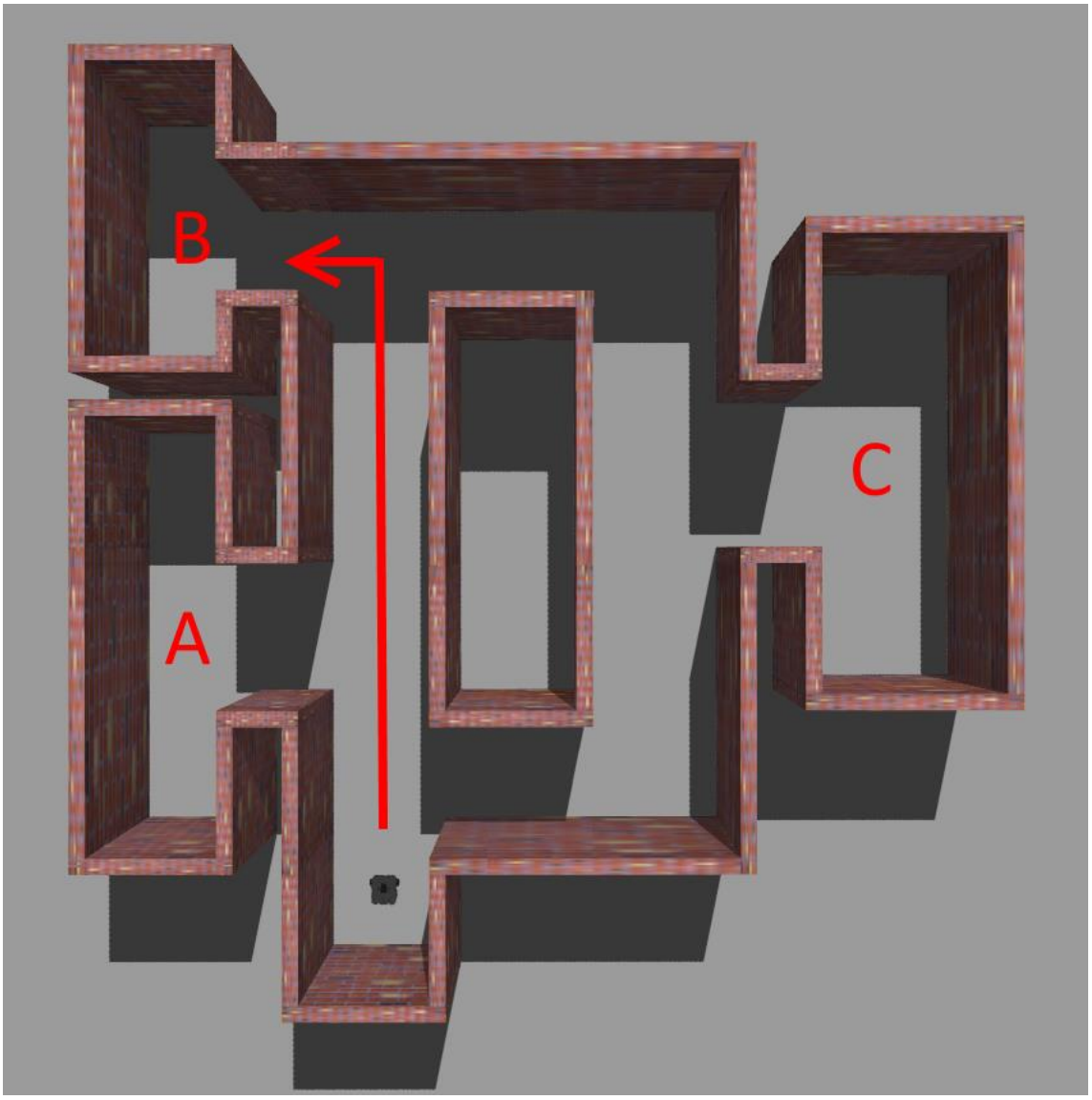
# Orders

## Experiment 1

Move the robot around the area it is in, experiment with telling the robot to move/stop and turn.

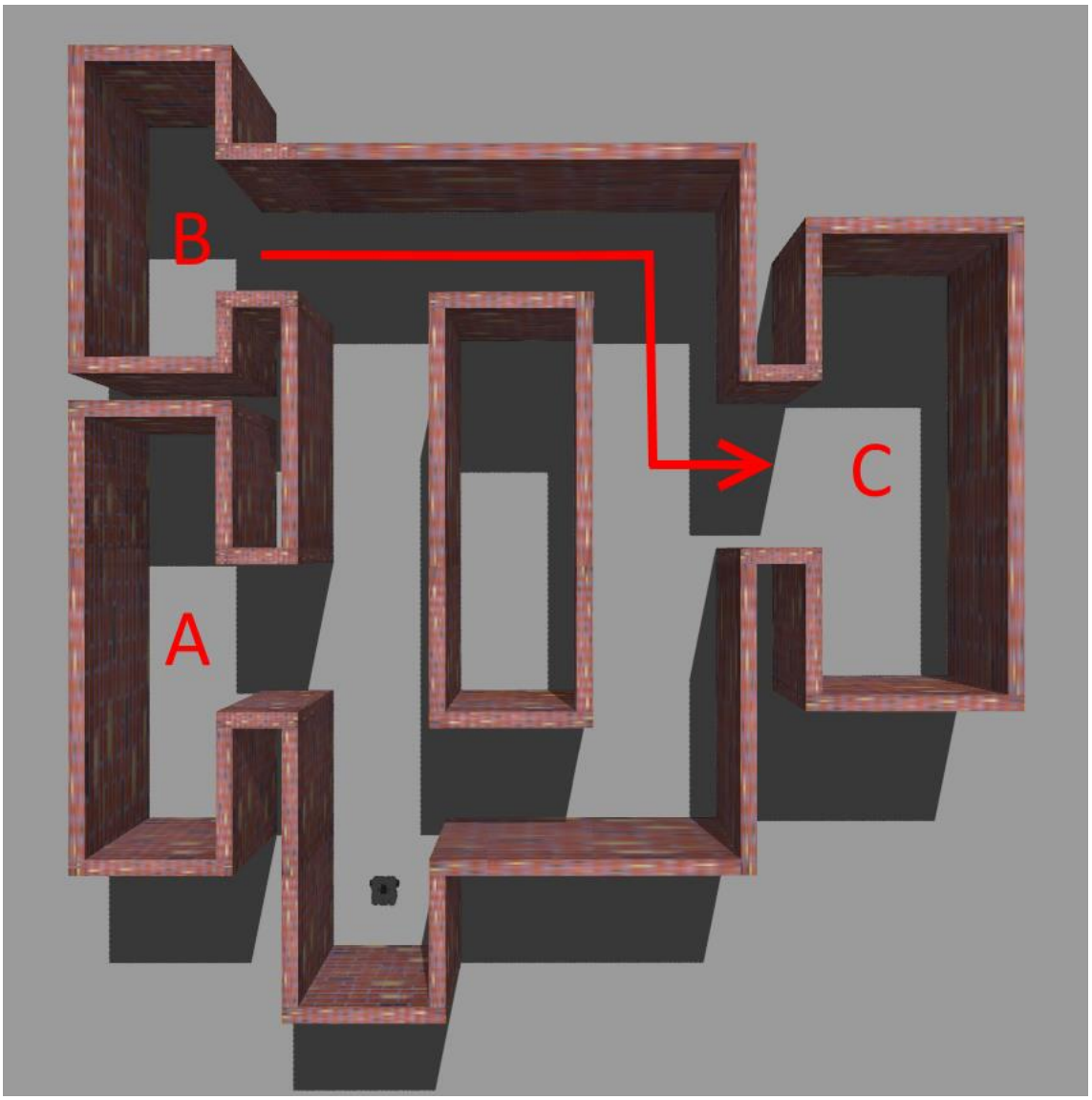
## Experiment 2

Give the robot directions to get to room B from the starting area, if possible, in one sentence.



# Experiment 3

Now that the robot is in room B give him directions to go to room C on the other side of the house, once more if possible, in a single sentence.



# Experiment 4

Return the robot to the starting area.