University of
Bedfordshire

Autonomous Tour Guide with Open-Source Software and the Internet of Things

AY19/20 - CIS017-3: Undergraduate Project

BSc (Hons) Artificial Intelligence and Robotics with a Professional Practice Year

Abstract

This research topic focuses on the development of an autonomous tour-guide that can operate in universities, museums and other public places of interest. The research discussed the current tour-guides such as Minerva and Rhino and the costs associated. Alongside the design of the navigation and path planning, human-robot-communication and testing methods. With the development of the hardware for the Turtlebot3 Waffle Pi and software using ROS and Python. The testing and results proved the project achieved it's aims and objectives of developing an autonomous tour-guide.

Author
Faisal Hoque
faisal.hoque@study.beds.ac.uk

Supervisor
Dr Dayou Li
dayou.li@beds.ac.uk

Dedicated to the loving memory of my Nan Alice Blackshaw, who always held me in high esteem, may she forever rest in peace.

Acknowledgements

I would like to thank my family for supporting me throughout this research project and allowing me to dedicate my time fully not just on this project, but also the last four years of university life.

Thank you to Susan Brandreth for supporting me throughout University life and giving me the opportunities to be at this point today. Without your help I would not have gained the programming skills I have today alongside other valuable skills.

Thank you Anisa for supporting me throughout the project and beyond and being there not just for your highly regarded creative skills but your gentle and kind loving heart.

I would also like to thank my supervisors Dayou Li and Renxi Qiu for helping me throughout this project and refining this research project into manageable goals. With their expertise and guidance, I have managed to complete this project with the desired outcome.

Thank you to Robert Keane and the Technician team for constantly being there and providing me with unlimited support when needed. Myself and the tour-guide are grateful you allowed the tour-guide to stay in the technicians office during a very rainy day.

Thank you to my fellow classmates and close friends throughout the last year and every year before that since we met. We had some good times during the course of this project, each giving me the drive and determination to continue on, hopefully we have many more moments to come.

I would also like to offer my gratitude to the library staff constantly keeping the library open and allowing me to access the plethora of resources when needed.

I would like to express my most sincere gratitude to everybody that assisted me with this project and I do apologise if I've missed anybody out. However, I truly believe everyone I've spoken to since initiating this project deserves to be acknowledged and thanked for their part.

Contents

Chapter 1

Introduction

1.1 Background

With a widespread awareness for software automation, academia and businesses are working to automate manual labour tasks in roles like a tour guide in places like museums, universities and homes. Despite the good progress in development for tour guides robots such as Minerva, Rhino, Sage, etc. The costs to develop these types of robots have traditionally been extremely costly ranging from around $20,000 to $50,000.

Even once developed the robots require large amounts of processing power; and are highly customised to their specific environment and would require further costs and maintenance for a dynamic environment. With further development in open-source software, hardware and the internet of things, the costs of a robot can be drastically decreased to merely $500 - $2,000 range.

1.2 Aims & Objectives

1.2.1 Aim

This research aims to develop an autonomous tour guide robot with readily available hardware and open-source software at a fraction of the costs of previous robots with similar functionalities.

1.2.2 Objectives

1. Review relevant literature on autonomous tour guides concerning robotic navigation, path planning, localisation and mapping.

2. Construct prototype with readily available open-source Turtlebot3 hardware from Robotis

3. Develop software that allows the constructed prototype to move autonomously in a predefined environment

4. Develop software that allows the robot and human to communicate with each other seamlessly

1.3 Research Methods

To gather sufficient research on this topic, a quantitative approach will be taken to analyse topics and case studies taken from other researchers that have also developed autonomous tour-guides. The research will also use a practical approach of experimenting with the developed artefact in a controlled environment.

1.4 Scope & Limitations

1.4.1 Scope

This report will cover the research of autonomous robotic tour-guides that can entertain, educate and interact with humans within public places like museums, commercial and academic environments. Traditional robots developed in the past have shown to work well within environments mentioned, though have come across issues with navigation for localisation, mapping and dynamic changes; human-robot interaction; and cost-effective resources.

1.4.2 Limitations

- Initial plan for experimentation was to use the newly built University of Bedfordshire STEM building on the Luton campus to conduct real-world tests. This test would provide relevant results on the viability of an autonomous tour-guide in a crowded environment and whether it is needed. Due to COVID-19 however, this test was not conducted as social distancing measures were in place nationwide. This means the research is lacking testing within a large environment and large crowds.

- The literature review will discuss available open-source hardware however, this research will prioritise on the Turtlebot3 Waffle Pi model for the open-source hardware as it is already owned by the University of Bedfordshire. Which means the research will have a lack of physical testing with other hardware that could potentially prove to be cheaper or better in some areas.

- Due to COVID-19 there will not be a web interface which would have allowed users to view the status of the tour-guide and authorised personnel to manually control the tour-guide.

- Due to COVID-19 the project plan went through three different variations to fulfil the main aim of the research. This meant not adding certain features such as: web interface to allow authorised personnel to view and control the tour-guide in real-time and utilising machine learning for autonomous movement.

Chapter 2

Literature Review

2.1 Related Work

This project is definitely not new and there has been a wide variety of autonomous robots developed for the purpose of guiding humans within museums, public places and homes. There are already museum guides out there such as two distinctive ones, Rhino [44] and Minerva [36], both robots were able to guide humans around a museum by using human-robot communications like buttons or foot tapping, etc. This section will go through the literature that already exists for autonomous tour-guides and discuss the features of these types of robots.

Out of the robots developed a majority of them were inspired by Rhino, developed in 1997 for the Deutches Museum Bonn (Germany), which interacted with roughly 2,000 people [44]. A more detailed investigation will be focused on Minerva, rather than Rhino or any other inspirations. As the techniques used for Minerva will be similar to the techniques deployed in this report, however at a fraction of the cost.

2.1.1 Minerva

Minerva is a second-generation robot that succeeds Rhino and was successfully deployed in the Smithsonian's National Museum of American History, interacting with roughly 50,000 people and traversing more than 44km. The choice to develop Minerva was to educate and entertain people in public places. While also addressing the issues such as safe navigation in unmodified and dynamic environments, short-term human-robot communications and keeping a virtual tele-presence [36, 37].

Figure 2.1: The physical appearance of Minerva [36]

Minerva utilises 20 distributed modules which can be broadly classified down into four groups: hardware interface modules, navigation modules, interaction modules and the web interface. Alongside each of those modules, Minerva utilises, probabilistic reasoning, learning and any-time algorithms within the software architecture to be able to make decisions, learn from experiences and have real-time reactions. The physical appearance of Minerva can be seen in fig 2.1.1 [36, 37].

In terms of localisation, Minerva first takes advantage of the sensors to generate an occupancy map, however due to large errors that can occur a camera is pointed at the ceiling to create a mosaic of the map also. Once the maps are created, Minerva can reference them to track its position, a complication however is people tend to frequently obstruct the robots sensors, for instance the lasers can be blocked by humans legs [36, 37].

Although the Minerva robot worked extremely well in a dynamic environment, the amount of processes and resources required for the robot were quite substantial. Minerva requires a great deal of processing power for the 20 asynchronous processes running at the same time, some processes running off multiple on-board and off-board PCs [36, 37, 42]. Of course this can lead to development costs being extremely high ranging from roughly $20,000 to $50,000. As majority of the components were not off the shelf purchasable components, the maintenance costs would also increase depending on the components [34].

2.2 Readily Available Hardware

After seeing the costs of previous robots developed and the magnitude of processes required to operate the robot. This section will go through the available robots that can be easily purchased with open-source software and hardware that allows for easy replacement and academic development.

Name	Hardware	Software	Cost
Turtlebot3 Burger	Raspberry Pi 3 B+, 360 LDS-01, OpenCR1.0, DYNAMIXEL XL430	Linux, ROS	$549
Turtlebot3 Waffle Pi	Rapberry Pi 3 B+, 360 LDS-01, Raspberry Pi Camera v2.1 OpenCR1.0, DYNAMIXEL XM430	Linux, ROS	$1,399
BIOLOID Robot Kit	BT-410, CM-530, DYNAMIXEL AX-12A, IR Sensor, Gyro	Behaviour Control Programmer, Motion Editor, Robot Terminal	$1,199
BIOLOID GP Robot Kit	BT-410, CM-530, DYNAMIXEL AX-12A & AX-18A, Gyro, Distance Measurement Sensor	RoboPlus Motion, RoboPlus Task, RoboPlus Manager	$2,199
ROBOTIS Engineer Kit 1	CM-550, 2XL430-W250-T Actuator	R+Task	$999

Table 2.1: List of open-source robots easily purchasable for academic and commercial purposes.

Turtlebot3 Burger & Turtlebot3 Waffle Pi [6], BIOLOID Robot Kit [26], BIOLOID GP Robot Kit [25], ROBOTIS Engineer Kit 1 [24].

Table 2.1 displays a list of readily available robot kits that can be purchased anywhere from $500 - $2,000 by anybody for academic or commercial purposes. Each kit has their own advantages and disadvantages, for instance the Turtlebot3 robot range is built up of modular plates that can be added and removed for specific purposes. The Turtlebot3's also have an on-board LIDAR which allows for detection obstacles to map a general area.

For the purpose of this project the Turtlebot3 Waffle Pi was chosen, the Waffle Pi is easily purchasable and modular, with open-source parts that can be easily replaced, removed and modified when needed.

2.3 Machine Learning Frameworks

Alongside the readily available open-source hardware, open-source software is also needed to be able to develop the logic of the mobile robot, in this case for the Turtlebot3 Waffle Pi. For the purpose of this project, machine learning will be used to achieve full autonomy and before that happens, an investigation into the machine learning libraries will be investigated.

Name	Programming Languages	Free?
Tensorflow	Python, JavaScript, C++, C, Java, Go	Yes
Pytorch	Python, C, C++	Yes
SciKit-Learn	Python	Yes
Theano	Python, C, C++	Yes
MATLAB	MATLAB, C, Fortran	No

Table 2.2: List of machine learning frameworks that can be used to develop a machine learning algorithm for autonomous movement [38, 19, 35, 39, 15]

Table 2.2 has a list of machine learning frameworks that can be used in developing an algorithm for complete autonomy. Out of those for this project Tensorflow is chosen because it is deploy-able on different code bases and has a large community backing. The community backing will allow for faster development in the algorithm as it will reduce the development time.

As a Turtlebot3 Waffle Pi will be used and the robot operating system is being used, open-source packages have already been developed to implement machine learning with a Turtlebot3 robot. The package utilises reinforcement learning with Deep Q-Learning, this means the software is more focused on how the actions are taken in a given environment to gain a cumulative reward [4].

Chapter 3

Artefact Design

The artefact is an autonomous mobile robot that can guide humans around a public place such as museums, universities and town centers, etc. In the case of the artefact a Turtlebot3 Waffle Pi will be used, which means the development will be primarily software based with a few hardware additions. This means the artefact design will be down to two primary software modules: Navigation and Path Planning and Human-Robot Communication.

Initially there was a third software module which was the Web Interface, however due to COVID-19 that was no longer possible as the plan had to be modified to allow the main aim to be achieved.

3.1 Navigation and Path Planning

The navigation and path planning of the turtlebot3 waffle pi will be based on the machine learning algorithm provided with the turtlebot3 in [4]. The first strategy will be to complete the four different difficulty models on the turtlebot3 manual for machine learning. This will give a better understanding in how the model works and for training and tweaking purposes. Then a manual map will be created by manually moving the turtlebot3 waffle pi through the STEM building first floor using tele-operations. Once the map is developed a 3D model of the map will be generated in Gazebo and deployed in the virtual environment for the turtlebot3 waffle pi.

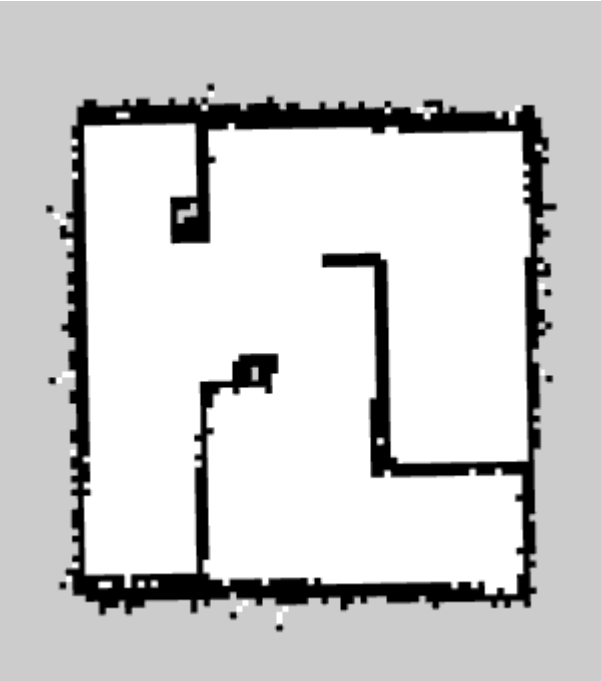


Figure 3.1: Occupany grid map [5]

Figure 3.1 shows a occupancy grid map (OGM) which is a two-dimensional space map, the grey cells indicated unknown areas, black space indicates occupied areas and white space indicates free space. This style of mapping is commonly used within the ROS community and is the outcome of mapping from a LIDAR sensor. A similar map will be generated for the STEM building using simultaneous localisation and mapping (SLAM), which will draw a map by estimating its current location in an arbitrary space [5]. Of course fig 3.1 is just an example of how a OGM looks, however the manually generated map for the STEM building will be similar.

Once the OGM is generated it can then be turned into a gazebo model which can be used in the virtual environment to train the machine learning algorithm to understand the STEM environment. The reason a virtual environment will be used is so the software can have unlimited number of trial runs in that environment. Which means it will be easier and faster to do testing and speed up development time.

In terms of deciding where the robot to go, the logic would be to have multiple routes programmed for the robot to go from A to B or A to Z, the default route would be to do a full tour around a default floor. The rest would be custom routes decided by users through the human-robot communication.

3.2 Human-Robot Communication

Humans and the robot will need to communicate with each other for the humans to indicate what kind of tour they would want and for the robot to establish specific rules if any and the to announce the routes that the robot can travel in. The main form of communication is to use voice communication utilising a speaker and microphone on the robot.

As it is a raspberry pi, the most easiest microphones and speakers for development purposes will be a USB microphone and speaker. The quality may be low to begin with however, to keep the development speed to time-line it makes more sense to use low quality components first and then to upgrade them as needed.

Once the speaker and microphone is connected a speech recognition program can be developed in python to listen for sounds and have a simple logic of if or switch statements to listen out for specific keywords which will trigger a response accordingly. This logic will then need to be more enhanced to be able to create custom routes going from A to B to Z then back down to C or D, according to the users preference. The simple logic will be: "Full Tour", and the robot will do a general tour from A to Z, though the path finding and navigation will be through the machine learning. The complex logic would then be something more like: "Show me D103, D101 and D102", this would be more difficult as the robot would need to know the fastest, best approach to this route. The robot would also need to make sure those locations exist in the first place.

3.3 Agile Testing

This project will be following an agile software development methodology, as will the artefact development itself. An agile software development methodology allows for continuous checks while development is still happening which means changes can be made instantly. As the methodology is an agile approach, the testing will also follow an agile approach. As the testing is done so will any changes that is needed to meet the requirements of the artefact.

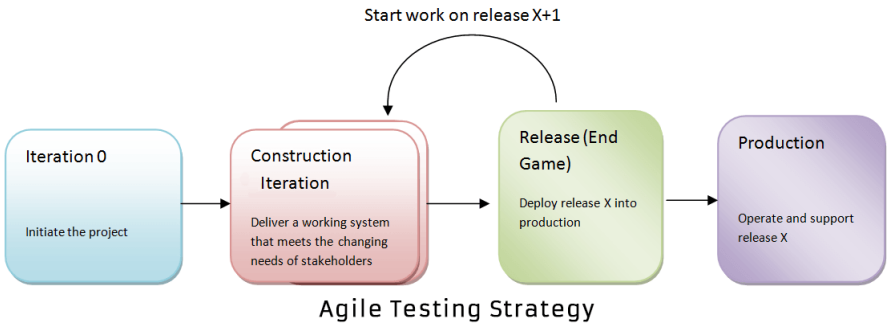


Figure 3.2: Life cycle of an agile testing strategy[7]

Figure 3.3 displays the life cycle of an agile testing approach. The initial idea is initiated and created, then it is continuously developed to meet the criteria of the requirements of the project and stakeholders. This construction iteration stage is continued until satisfactory goals have been met to deploy the product into production. Once deployed in production, the project is maintained and supported continuously as needed.

Chapter 4

Artefact Development

4.1 Assembling Hardware

For the assembling of the artefact a range of different components were used, from the Turtlebot3 Waffle Pi components and two additional components, a speaker and microphone, both required for speech to text and text to speech. The following sections will discuss the components, the relevance and requirement of them and then the process of assembling the tour-guide.

4.1.1 Turtlebot3 Waffle Pi

The Turtlebot3 Waffle Pi model is a collection of open-source components packaged together by ROBOTIS, which can then be assembled into the Turtlebot3 Waffle Pi. This package comes with a single board computer, embedded board, actuators, sensors for different purposes and building blocks for a dynamic structure. ROBOTIS advertises the image shown in 4.1.1, as the final outcome after assembling the components.

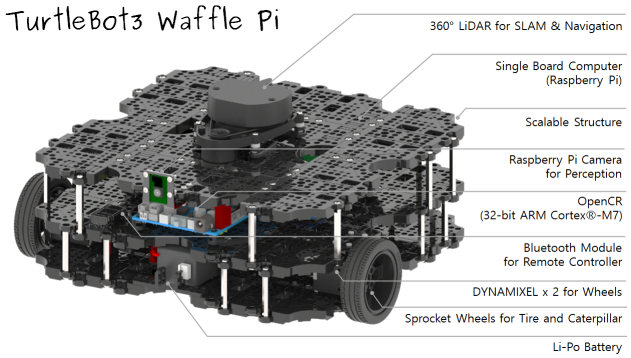


Figure 4.1: Turtlebot3 Waffle Pi with all the components fully assembled [28]

The following list contains the individual components in the Turtlebot3 Waffle Pi package and an explanation for the requirement of each component. More information for each component can also be found in the Turtlebot3 Specifications manual [28].

- Raspberry Pi 3 Model B+ - this is the only on board computer that controls each individual component on the structure as listed below [17]. Shown in figure A.1.1
- LDS-01 - this is the 360 degrees LIDAR that will allow for detection of obstacles using infrared lasers. This is crucial to allow the tour-guide to know where it can and can't move and will be used with SLAM for navigation purposes [21]. Shown in figure A.1.1
- Raspberry Pi Camera Module V2 - this camera allows the tour-guide to see obstacles in an image perspective. Currently it is not being used, however will be discussed further for future development [16]. Shown in figure A.1.1
- OpenCR1.0 - this board is the main controller of the tour-guide which controls the motors and has additional sensors for detecting current geometry, etc [23]. Shown in figure A.1.1
- XM430 - two of these are the motors which allows for movement of the tour-guide and is controlled by the OpenCR board [33]. Shown in figure A.1.1
- BT-410 Set - this includes a master and slave bluetooth module which enables for wireless control of the tour-guide [20].
- LI-PO Battery 11.1v LB-012 - this is the main power source of the tour-guide [22]. Shown in figure A.1.1
- TB3 Waffle Plate IPL-01 - these plates make up the structure and body of the tour-guide and allows the ability to position components dynamically [31]. Shown in figure A.1.1
- TB3 Wheel Tire Set ISW-01 - the wheel and tires are attached to the motors for movement of the tour-guide [32]. Shown in figure A.1.1
- TB3 Ball Caster - A01 - two of these makeup the back movement of the tour-guide [29]. Shown in figure A.1.1
- TB3 PCB Support IBB-01 - these allow positioning of components like the raspberry pi and opencr boards to be properly placed [30]. Shown in figure A.1.1

4.1.2 Human-Robot Communication

The main method of communication between people and the tour-guide will be through speech synthesis and voice recognition. This will allow for a more natural and seamless interaction between both parties. The Turtlebot3 Waffle Pi's package however, does not come with a microphone or speaker. The Raspberry Pi does in fact support specific microphones and speakers. The following additional components were purchased and added onto the tour-guide.

UKHONK Portable USB Speaker

The requirement for the speaker needed to be small enough to fit comfortably within the chassis of the Turtlebot3 Waffle Pi and be loud enough to be heard from the ground. The UKHONK devices offered three different variations on the model. The first being a small oval shaped design as shown in figure A.1.2; the second being a more rectangular base, slightly larger than the first as shown in figure 4.1.2; and the third also being rectangular but much larger as shown in figure A.1.2.

Figure 4.2: UKHONK Portable USB Speaker Model B [40]

The decision was to choose the second model, as it was small (3.7x1.8x1.1 inches) enough to fit into the chassis comfortably and was loud enough with a speak power of 3W. A USB speaker was chosen over an audio jack was because the USB speaker would be plug and play and easier to use.

UNOOE Omnidirectional Microphone

The microphone was the more difficult to decide on which one to get, as there are a plethora of microphone polar patterns [3] to choose from and in terms of which would be the best to capture for the tour-guide was a difficult decision to make. However it was down to either a unidirectional microphone, which picks up sound from a single direction the microphone is pointing. And an omnidirectional microphone which picks up sound from all directions.

The decision was to choose an omnidirectional microphone as people may be behind the microphone trying to talk to the tour-guide so a unidirectional microphone would not work in this use case. It was difficult to find one that would come quickly due to the current situation with COVID-19, so it was decided to use the UNOOE omndirectional microphone as shown in figure 4.1.2.

Figure 4.3: UNOOE Omnidirectional Microphone [41]

4.1.3 Combining Components

Once all the components were sorted out, it was time to assemble them together, the first step was to assemble the Turtlebot3 Waffle Pi as it came with an instruction manual and would give a clear indication where the speaker and microphone would fit. The Turtlebot3 Waffle Pi is built in three layers, the first layer is the motors and battery; the second layer is the more important part as it contains the Raspberry Pi, OpenCR board, bluetooth module and an adapter for the LIDAR as shown in figure 4.1.3; the third layer consists of the LIDAR module and the Raspberry Pi v2 Camera module as shown in figure A.1.3.

Figure 4.4: Assembling Turtlebot3 Waffle Pi second layer

It is very important to install the software in the Raspberry Pi once the second layer is installed, or else the third layer would need to be removed again to put the memory card back into the Raspberry Pi. A few issues did occur when assembling the Turtlebot3 Waffle Pi, due to lack of experience at the time and not following the instructions properly.

- The Turtlebot3 Waffle Pi was fully assembled which took roughly 6 hours to complete. However once assembled the realisation was the software was not working and this was because the software was not properly installed and updated. A monitor with a HDMI cable is required to operate the Raspbian operating system with a mice and keyboard. The WiFi would need to be configured, the software was already configured, just needed updating.

- The motors were put on the opposite sides, which meant the Turtlebot3 Waffle Pi would move backwards if told to go forward and vice versa. It is important to double check the placement of the motors correctly as shown in the instruction manual.

Once the Turtlebot3 Waffle Pi was fully assembled, it was time to place the speakers and microphone in and plug them in. It was easy to put the microphone and speaker in as they were USB. The placement was done using zip tie cables as it meant the robot didn't need to be taken apart once constructed fully. The placement of the speaker and microphone can be seen in figure A.1.3.

4.2 Developing Software

The software consists of two packages, the first is to allow the tour-guide to move autonomously to a predefined location and the second is the allow the tour-guide to communicate with people. This section will discuss both packages and the process in developing the code and why that strategy was chosen.

4.2.1 Autonomous Movement

The initial plan for autonomous movement was to use the Machine Learning library provided by ROBOTIS in ROS, which used reinforcement learning with DQN. However due to COVID-19 and not being able to have access to a strong enough computer to create the simulations for the machine learning environment. The decision was later made to use the Navigation library provided by ROBOTIS [27]. This means a map that contains geometry information is required for it to work. The first step was to create a map of the STEM building using SLAM, which can be seen in figure A.2.1. However because of COVID-19 the STEM building was no longer accessible so a map of a hallway in a home was also created as depicted in figure 4.2.1.

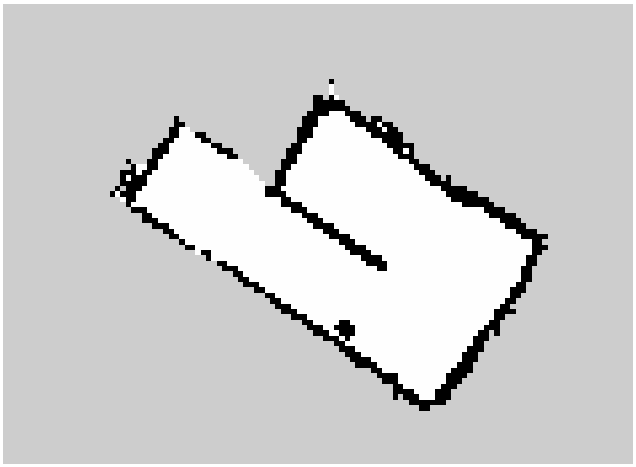


Figure 4.5: SLAM Mapping of home corridor

Once the maps were created, it was important to understand how the navigation package worked. Once the package was correctly launched, a GUI appears as shown in figure 4.2.1. Once loaded the map will need to be correctly positioned using the 2D Pose Estimate button so the map is pointing the same direction as the tour-guide.

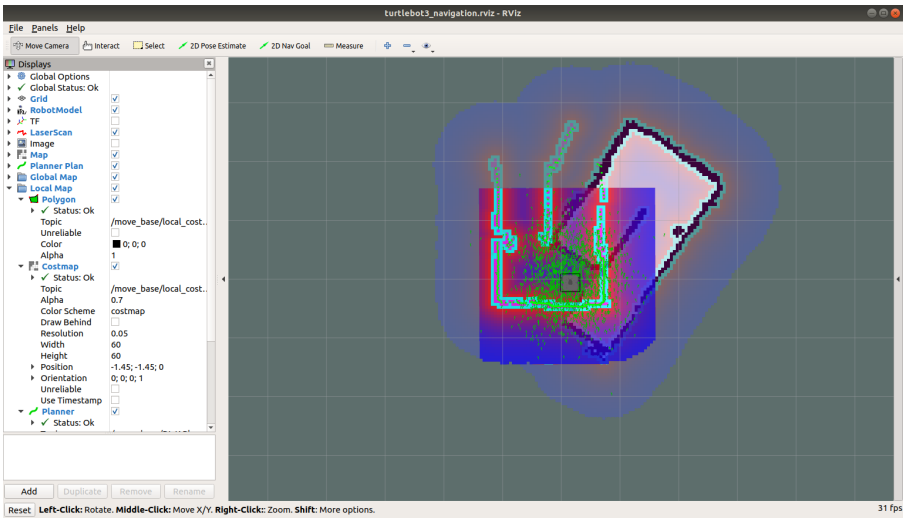


Figure 4.6: Result of running Navigation node first time, with incorrect pose

Once the pose is corrected as shown in figure A.2.1, the 2D Nav Goal button was then used to determine which ROS node is controlling the movement. It was apparent the Navigation package was using the /move_base [14] node to move the tour-guide in the direction required. The next step was to create a ROS node using Python3 that would control the tour-guide using the /move_base node. As shown in A.2.2 a simple action client was created to interact with the /move_base package, as demonstrated in figures A.2.1 & A.2.1. Once the method of figuring out how to move the tour-guide dynamically was figured out, the next step was to create a data set of notable locations that the tour-guide can tour around.

For this the /move_base/goal node was echoed to see where the tour-guide is currently moving and /acml_pose was used to see the current position of the tour-guide. This meant the tour-guide can be positioned in front of a notable position in the map and the /acml_pose node can be called to see the current position. The information was then collected into a JSON file for easy transfer and readability as shown in figure A.2.2. The code was then refined into a more easily accessible method which can be reused throughout the implementation of the tour-guide. The method is called robot_move() and can be seen in figure A.2.2 on line 80.

4.2.2 Human-Robot Communication

The human-robot communication package is called robot_speech and condensed into two Python2 ROS services that is run on the tour-guides' single board computer. The first service is the speak.py file which can be seen in figure A.2.2. It utilises the pyttsx3 library for Python, which enables the text to speech

feature [2]. The method was initially developed on windows, then transferred to Ubuntu to be converted into a service, which was easy to do as only the requests variable was required and a service node created. The method allows for converting of text to speech which can then be easier to interpret for people.

The second service is the listen.py file which utilises the SpeechRecognition Python library [45] to convert speech to text as shown in figure A.2.2. This service utilises the microphone on the tour-guide and returns a string object in JSON format. This means the string object can be easily converted to a JSON object using the Python json library [18]. The SpeechRecognition library supports eight different speech recognition engines such as the following:

- CMU Sphinx - Free to use and works offline, however requires large amounts of data for training and is a much larger package to install.
- Google Speech Recognition - Free to use with the generic API key provided, however no longer possible to get additional quotas for the API key, forces you to use Google Cloud Speech API instead.
- Google Cloud Speech API - Costs $0.009 / 15 seconds, after the first 60 minutes.
- Wit.ai - Completely free indefinitely, requires some training but uses already trained data from other publicly available projects.
- Microsoft Bing Voice Recognition - Five hours free per month, costs $1 per hour afterwards.
- Houndify API - Requires an account and has a credit system which needs to be refilled with money every time it depletes.
- IBM Speech to Text - Free for first 500 minutes per month, costs vary after specific tiers are reached but first tier is $0.02 per minute.
- Snowboy Hotword Detection - Free to use for personal purposes, however costs for commercial use and is shutting down the services after 31st December 2020.

The initial experiments were with Google Speech Recognition, however after realising there's a potential of reaching over the limit and not being able to use the tour-guide afterwards, new alternatives were used. CMU Sphinx and Snowboy were experimented with, but Wit.ai was chosen in the end due it being completely free unless doing 1 request per second, which is impossible with this project and because it was a combination of Snowboy and CMU Sphinx.

This project is using Wit.ai for keyword identification and also for sentence recognition. When it hears a specific keyword like "toilet", in any sentence it will trigger a movement keyword which the tour-guide can respond to. For the sentence recognition saying: "Give me a full tour", would be the same as: "Show me around the building", both sentences will trigger a "tour" intent.

4.2.3 Combining Software Packages

To combine the packages into one main software, two parts were created, the first is a utility package which contains methods that help read the JSON data file and any mathematical calculations; the second being the main file that contains the loop which controls the hearing, speech, and movement of the tour-guide.

The utility package can be seen in figure A.2.2, the first method on line 9 is get_map_nodes(), which retrieves the JSON data and converts it into a Python array of objects so it can be easily accessed. This method is called quite frequently in the run_robot.py class, therefore it needed to be a re-useable method.

The second method on line 16 is the xy_dist() method, which calculates the distance between two x, y points. The experimentation section will show this in further use, and the evaluation will explain it in more detail. However it is required to identify the shortest distance between two points.

The third method on line 31 get_tour_nodes(), takes in the map_nodes object and the current location to determine the closest locations to start showing first. This will return a Python array similar to map_nodes which an additional parameter to the object in the array called "distance", which shows how far away it is from the previous object in the array. An example of this can be seen in the experimentation section in figure 5.1.3.

The second package is the run_robot.py file which can be seen in figure A.2.2. This file controls the main loop and all the actions the tour-guide can perform, from listening, speaking and moving. The movement method can be seen on line 80, robot_move(), which takes in the map node that contains the geometry information for that specific map. This creates a simple action client that communicates with the /move_base node and moves the tour-guide to the desired pose (x, y) and orientation.

The speak method can be found on line 103 robot_speak(), this creates a request to the speak service created in figure A.2.2. Any string value can be provided in the parameter and as long as the service is running on the tour-guide, it will be spoken through the speakers.

The listen method is on line 112 robot_listen(), and this also creates a request but this time sends it to the listen service as shown in figure A.2.2. This method waits for the service to listen through the microphone on the tour-guide and once a value is sent back in string format. This method will convert the returned value into a Python object that gives easy access to the text spoken and the potential intent Wit.ai identified.

There is also a method that retrieves the current location on line 132 get_amcl-_pose() and assigns it to a global parameter of amcl_pose using the assign_amcl-_pose() method on line 124. This method is required to use the get_tour_nodes() method in the utility package as shown in figure A.2.2, the method will return an object with x, y values respectively.

The main lmethod is on line 25 listen_to_robot() as shown in figure A.2.2, this method is a series of if statements that checks the intent and does a specific logic for each one. For instance line 31 in figure A.2.2 shows an if statement that

checks if the intent value is "tour" and if so then confirm with the user, which then checks if they agree/disagree. Then the tour-guide will loop through each node within tour_nodes and move to that location respectively and provide a brief description to each. This style repeats itself for the keyword section too, however the difference here is once a keyword is detected. The keyword is then checked to see if it exists in the map_nodes JSON file and if it is then it will move to that node, as shown on line 61 in figure A.2.2.

Chapter 5

Testing, Results & Evaluation

This section will reiterate through the objectives and specify the individual goal for each and test the features to see if the objectives have been achieved and discuss the evaluation on the results. There were three objectives to achieve with the tour-guide itself, as listed below:

- Speech synthesis & recognition - This objective is to see if the tour-guide can speak, which will allow effective communication to users. And whether the tour-guide can actually listen to users as well.
- Full tour - This objective requires the tour-guide to be able to loop through tour-able locations and move to them.
- Move to specific location - This objective needs to be able to listen to a specific keyword spoken by the user and move to that location if exists.

5.1 Speech Synthesis & Recognition

This speech synthesis & recognition is an important feature for the tour-guide, without it the tour-guide would not be able to communicate with the users. This would make the other features such as giving full tours and moving to specific locations redundant. That is because there would be no way of conveying instructions to the tour-guide. This section will cover the goals for this feature and testing whether they work as intended and discussing the issues in more detail.

5.1.1 Goals

- Be able to speak any form of sentence through the speakers for proper communication with the user.

- Be able to listen to a phrase and determine the intentions of that phrase and associate it with multiple phrases that could have the same intention.
- Be able to listen to a specific keyword and determine the intent of that keyword and react accordingly.

5.1.2 Speech Synthesis

The speech synthesis feature uses the pyttsx3 library [2], which makes it much easier to enable voice from a speaker. As the developed code is a separate node and can be run independently from the main node. It can be tested individually to see if the software is working as intended, as shown in figure 5.1.2.

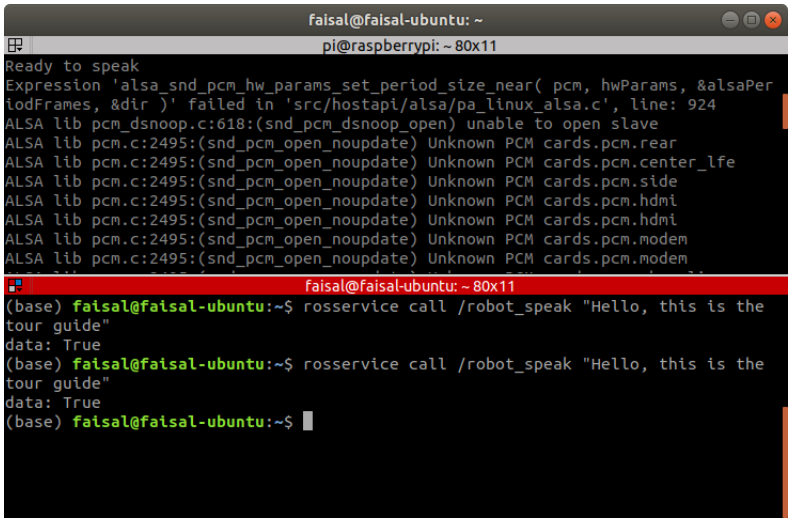


Figure 5.1: Command line showing the speech node

To reproduce this follow the steps below:

1. Master machine: roscore
2. Turtlebot3 machine: rosrun robot_speech speak.py
3. Master machine: rosservice call /robot_speak "Test string"
4. The "Test string" can be replaced with anything but must be wrapped in double quotes to indicate a String object.

Issues Encountered

The initial package was developed on the windows operating system which has only two different variations in voice, one for male and one for female, both are English. However once moved over to a Linux operating system, there are now

over 200+ voices for different languages, i.e. English, Scottish, Arabic, Bengali, etc. This caused an initial issue as the windows version of the program had the first index of voices selected which indicated female. However on Linux, this was Albanian, and once spoken created confusion and cost a few hours of development time until realising their are multiple languages. It was quickly resolved by looping through the voices and selecting the English one specifically.

Evaluation

This feature was easy to implement as there is a lot of documentation to specify how it works. The only real difficulty came when moving the code from windows to Linux, however that was just semantic error rather than syntax. The goal of being able to convert text to speech was achieved and can be demonstrated in the following video [13], as pictures cannot do sound demonstrations any justice.

5.1.3 Speech Recognition

The speech recognition feature utilises the SpeechRecognition Python library [45] and specifically the Wit.ai [43] API. The speech recognition package is also an independent node which will run on it's own and can be tested independently from the main tour-guide node, as shown in figure 5.1.3.

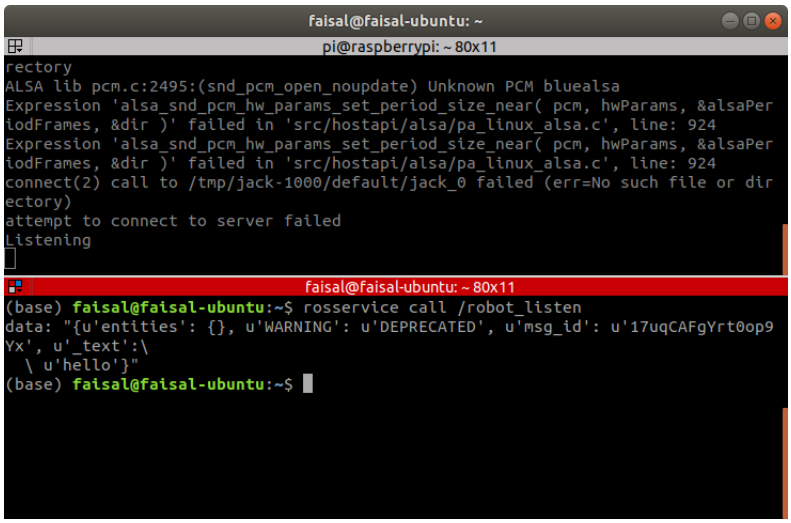


Figure 5.2: Command line showing the listening node

The node will listen through an available microphone and once it hears any words, it will send a String of a JSON object over to the client that called the service. Wit.ai API will try to identify the spoken word/sentence and try to associate a keyword or intent to that, in the case of this test it was just saying "hello" which has no intent. However if the sentence "show me around"

is spoken, then the intent of "tour" will be returned to indicate the user wants a tour. Follow the steps below to reproduce this test. As the service takes no parameters, you do not need to provide anything and can leave the parameter empty as shown below.

1. Master machine: roscore
2. Turtlebot3 machine: rosrun robot_speech listen.py
3. Machine machine: rosservice call /robot_listen

Issues Encountered

- The initial program used the Google Speech Recognition API, however this API was limited to a certain amount of requests per key and you could not request a higher request limit. The other solutions were to use Google Cloud Speech API, Microsoft Bing Voice Recognition, Houndify API, IBM Speech to Text. However all of these had one thing in common, where payment would be required after the free usage was used up. As the research is mainly in using open-source software, it wouldn't make sense to use those. The other free to use options were CMU Sphinx, Wit.ai and Snowboy Hotword Detection. CMU Sphinx and Snowboy could be performed offline. However CMU Sphinx required thousands of training data and a large file, not suitable for a raspberry pi. Development for Snowboy on the other hand will be terminated after 31st of December 2020. Wit.ai was a combination of both Snowboy and CMU Sphinx, as it allowed for keyword detection too and returned the full spoken text.
- The USB microphone used had issues with Linux as the voice would slightly cut out, making certain words unintelligible. For instance when spoken: "give me a full tour", Wit.ai would think it said: "give me a fool", it still correctly identified it as a "tour", however this gives false training data. A test would need to be done with a audio jack microphone to see if there are better results with clear voice recordings.
- There are times when the program will hear nothing and this will throw an error and break the program. This fix to this was to implement a try, except block which catches the error and just returns an empty object as shown in figure A.2.2.
- The program throws warnings when it tries to access the microphone initially, however these are just warnings stating it couldn't connect to an audio device. This is because the library loops through available devices and tries to connect to the first microphone available. This does not break the code, however can be fixed by specifying the microphone index like so: Microphone(device_index=5), however the index can be different to other users, so it was left alone.

Evaluation

The implementation was fairly simple as there was a lot of documentation and tutorials in developing this feature. The only real issues were in deciding which microphone to use and the API service. However after proper research both can be chosen relatively quickly and this research should give a clear indication why Wit.ai was used. In terms of the microphone, a USB microphone isn't the most clear for linux operating systems. This could be because some settings need to be tweaked, as the same USB microphones work fine with windows. The goal of being able to listen to a phrase and produce the correct intent was achieved. A demo of this feature can be seen in the following video [12].

5.2 Autonomous Movement

This feature is the main feature the tour-guide exists, which is to show users that are entering buildings like museums, universities, etc and be able to show them around and any specific locations a user would want to view. This section will cover the goals for this feature and testing whether they work as intended and discussing the issues in more detail.

5.2.1 Goals

- React accordingly to the returned intent, whether it is keyword or phrase.
- If the intent is "tour" then navigate through all tour-able locations and speak a brief description for each.
- If the intent is a keyword to a specific location like "toilet", then move in that direction.

5.2.2 Full Tour

This section will experiment the full tour feature which should loop through all the tour-able locations within the map_nodes.json file as shown in A.2.2.

First Experiment

The first experiment was to loop through every single location within the map_nodes.json file regardless of whether it is a tour-able location or not. An example of this has been shown in figure 5.2.2 to demonstrate how the full tour feature worked initially.

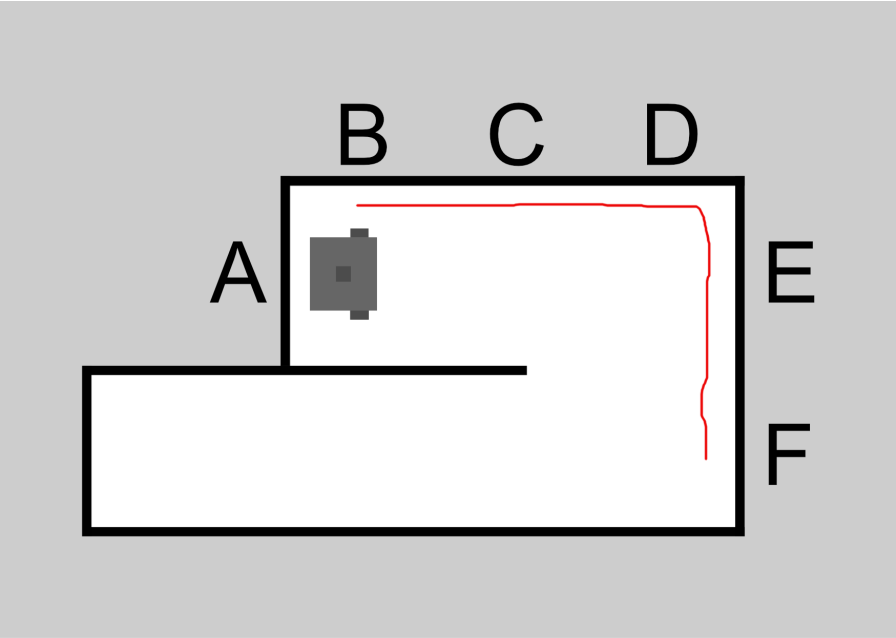


Figure 5.3: Shows a diagram of the tour-guide moving from locations A to F

As shown in figure 5.2.2, the tour-guide will move to every location, A, B, C, D, E & F. However some of those locations may not be relevant to show within a full tour. For example in a museum, the tour-guide may need to show relevant exhibitions and miss out things like the toilet, staircase, storage rooms etc. However a tour-guide within a home for sale, may need to show the toilet, storage rooms, etc.

To work around this issue, a new variable was added into the JSON file for each object known as "in_tour", which is a boolean variable of true or false. The user should change this value to true depending on whether the location should be shown in the tour. This change means the tour-guide will now travel to A, B & D and stop at D, as shown in figure B.1.1. A demo video of this can also be seen here [8].

Second Experiment

The second experiment was to test to see how the tour-guide would operate if the full tour was requested on the opposite side of the map. As shown in figure 5.2.2, the tour-guide will go from A to F that is because it starts at A. However, what happens if the tour-guide starts at F first, as shown in figure 5.2.2, the tour-guide ends up going for location A first then loops back to F.

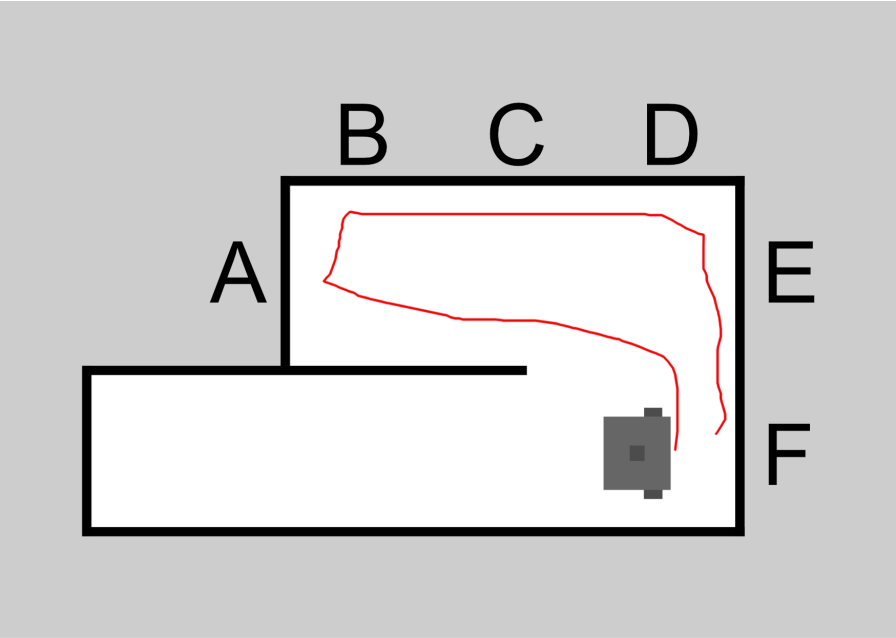


Figure 5.4: Shows a diagram of the tour-guide starting at F, looping through and ending back at F

As figure 5.2.2 shows, the tour-guide will start at F then go to A, B, C, D, E and back to F. However it should actually start from F and end at A, since F is the closest location. To fix this, a calculation needed to be made to determine the closest location to the tour-guide and the next closest to that location, etc. As shown in figure A.2.2 line 21, get_tour_nodes(), will loop through map_nodes.json and return the nodes that have "in_tour" set to true and the begins with the closest location to the tour-guide. The calculation to determine the distance is shown below and the code variant is in figure A.2.2 line 16.

$$D_2 = X_2 + Y_2 \tag{5.1}$$

Get_tour_nodes() is a recursive method, the reason it's recursive is so it can start with the current location of the tour-guide. But once it finds the closest location, the new current location becomes the closest location. This then loops through until it finally organises the tour nodes correctly in the order they should be toured. This can be seen in figure 5.2.2 where the tour-guide starts at F and ends on A.

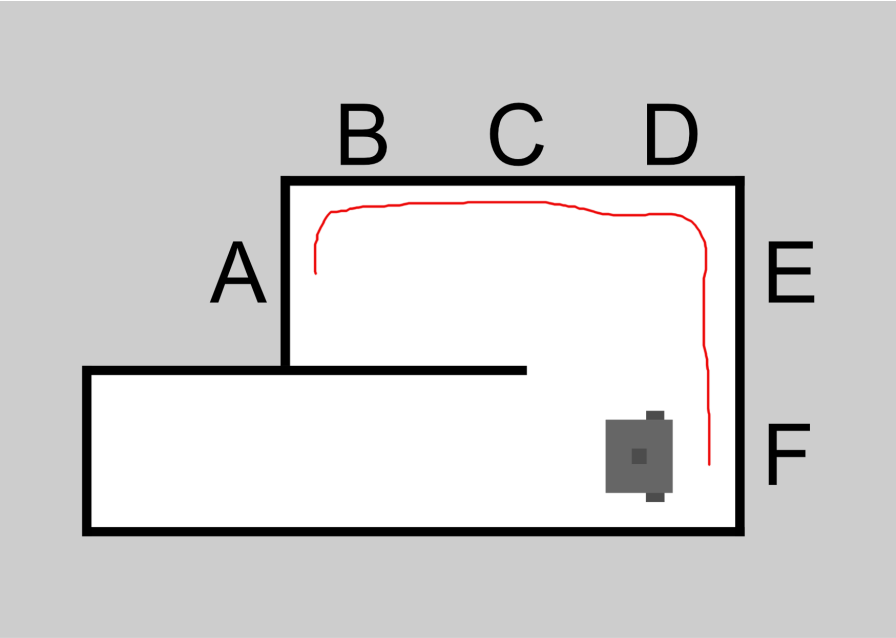


Figure 5.5: Shows a diagram of the tour-guide starting at F and ending at A

This can also be seen in figure B.1.1, where the tour-guide starts at C then goes B, A, D, E and ends on F. However one thing to note is that it may not always be the fastest path. As shown in figure B.1.1, the tour-guide starts at E, then D, C, B, A and ends on F. However it should probably start at E, then F, D, C, B and end on A, as this route would be faster. A video demoing the full tour in reverse can be seen here [9].

Third Experiment

The third experiment was to check if the tour-guide would stop or continue moving if a person came in front of the tour-guide. Due to COVID-19 it was not possible to record this interaction properly. However, the tour-guide tries to replan to move around the person if they get in the way. The interaction however is quite sensitive and if the object is too close, the tour-guide can get stuck in a loop trying to find a new plan. This interaction can be seen in the following video [10], where the tour-guide got too close to the door and got stuck trying to find a new plan. This was quickly fixed as shown in the video by moving the tour-guide slightly away from the door.

5.2.3 Move To Specific Location

This feature was more of an extra addition which allows users to locate a specific area, for example a common one would be: "where is the bathroom?". The tour-guide can then take the user directly to the bathroom.

First Experiment

The first experiment was to see if the tour-guide will respond to specific keywords like "toilet", "small room", "stairs", etc. In the case of explanation, the rooms in the diagrams will be lettered from A to F. The good thing about keyword identification is, it will only move to that location if a keyword is said. Which means, the keyword "toilet" can be said in any phrase such as: "where is the toilet", or "show me the toilet", etc. Once said the tour-guide will move towards the toilet, as shown in figure 5.2.3.

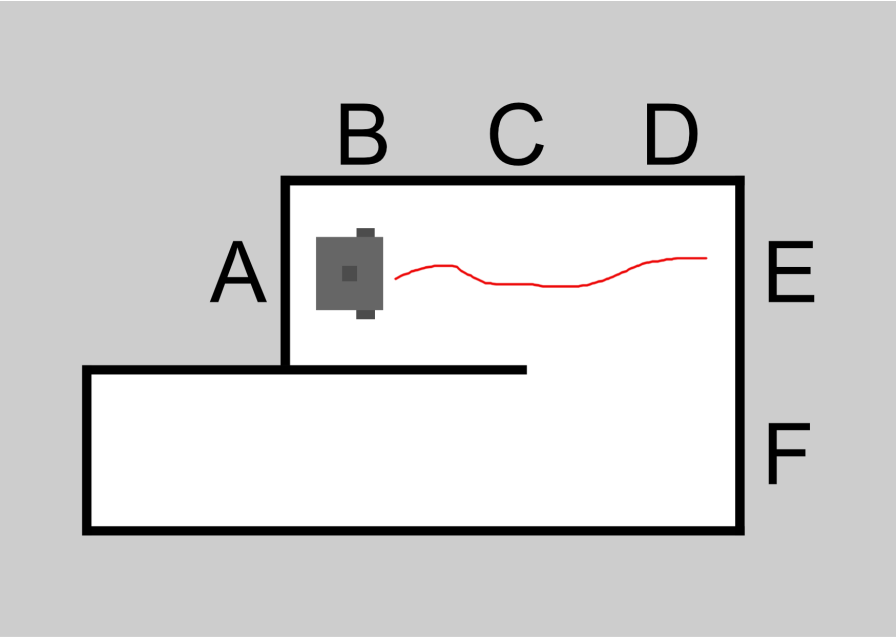


Figure 5.6: Shows a diagram of the tour-guide moving from A to E

The issue with this approach was the tour-guide could only identify the location of toilet as the keyword of "toilet". However some people may call it bathroom, loo or rest stop, etc. To allow for this additional feature, another variable was added to the map_nodes.json file. This time it was "keywords" as shown in figure A.2.2 line 5. The "keywords" variable is an array that can contain multiple keywords to identify that particular location. This level of adaptability means the tour-guide can now move to the same location with

multiple keywords. A video demonstrating the tour-guide moving to a specific location can be seen here [11].

5.2.4 Issues Encountered

- One issue when speaking to the tour-guide is there is no light indicating it is listening or when to actually speak. This means the user may not speak at the right time, which means the user may repeat themselves. This can be fixed by adding an LED to indicate when to talk.

- When speaking to the tour-guide to request either a full tour or show specific location. The tour-guide's communication is one-side, which means it will confirm if the user wants a full tour. However if the user does not respond in time, then the tour-guide will cancel all instructions. A fix to this would be to reconfirm at least two or three times to give the user a chance to respond.

5.2.5 Evaluation

The implementation of the autonomous movement feature was simple enough to do, as it was a matter of combining packages that already exist to operate together. The only new package that was needed was the robot_speech package, which allowed for voice synthesis and recognition on the Turtlebot3 Waffle Pi. The rest of the logic as shown in figure A.2.2, is primarily if/else statements which checks to see what a users intent was and to respond accordingly.

The biggest difficulty throughout the development of this feature was COVID-19, as the tour-guide was initially meant to tour around the STEM building for University of Bedfordshire. As there are lock down restrictions, this meant the autonomous feature could not be tested in the STEM building.

The experimentation's meant features that did not work could be fixed and enhanced through testing and further development. This meant people can operate the tour-guide more seamlessly and easier. Overall the initial goals set for the autonomous movement was achieved, which means the overall objective was also completed and contributed further to the aim of this project.

Chapter 6

Conclusion & Future Work

6.1 Recommendations

While developing this project a few extra notes were taken down which could potentially improve the tour-guide, however would require testing of course as they were not tested in this research topic.

- For the speaker and microphone, Amazon Alexa could be used instead of two individual usb mic/speaker. This could potentially improve the quality of speech recognition drastically, as Alexa has a larger training base. Alexa Skills Kit [1] can be used to quickly develop a script that listens to the user and trigger a response accordingly. The only drawback is, the code would need to be developed, as the robot_speech package cannot be used for Alexa.
- When building the Turtlebot3 Waffle Pi, it is important to install all required software before assembling. As once assembled, it will be difficult to put in the memory card and stages will need to be disassembled.
- The speech recognition API Wit.ai can be further improved by training it further with different scenarios and accents, so it can understand better.

6.2 Future Work

This project can be further developed with additional improvements to make the tour-guide more effective and efficient at touring locations.

- The current system of adding/removing locations to the tour-guide is manual, but can be further improved by developing a GUI, whether that is a desktop, mobile or web application. This application could also view the tour-guide in real-time on a map as it moves and control it externally. All these functionalities already exist, only a GUI would be needed to control them.

- Instead of using the navigation package provided by ROBOTIS, the machine learning package could be used. This would allow for learning a new environment, rather than passing in a predefined map. However, this research topic would be a project in itself and should focus primarily on the machine learning aspect of learning new environments.
- Object detection packages can be added to identify people as it is touring around so it can communicate with them even more effectively. The tour-guide can activate the listening software once it identifies people are in the vicinity, etc.

This research topic can also be converted into other purposes than just a tour-guide, but also as a patrolling robot to patrol the streets, especially during times like COVID-19. A robot that can patrol the streets could maintain lock down measurements safer than police officers.

This research topic could also be used as a base to develop the University of Bedfordshires Dalek, which currently has motors and is move-able. It just doesn't have a single board computer to act as a brain. The packages developed here, could be extended to give life to the Dalek and also turn that into a tour-guide.

6.3 Conclusion

This project focused on the research and development of an autonomous tour-guide that can operate in universities, museums and other public places of interest. The introduction presented the aims and objectives this project will attempt to achieve. The aim being the development of an autonomous tour-guide with readily available open-source hardware and software. The objectives in summary were to collect relevant research; construct the turtlebot3 robot; achieve autonomous movement and software to communicate with people seamlessly. A quantitative approach was taken for the research of this project. The scope focused on the research of autonomous robot tour-guides. The literature review was limited to specifically Turtlebot3 Waffle Pi and the project was limited due to COVID-19.

The literature review covered the related works of Minerva, Rhino and similar tour-guides. Other readily available hardware than Turtlebot3 Waffle Pi was also discussed with the costs, advantages etc. Alongside the machine learning frameworks available and could be used, however due to COVID-19 the chice of machine learning was changed to navigation packages as discussed in the development section.

The artefact design discussed the plan for development of the navigation and path planning and which algorithms will be used. Alongside the human-robot communication and the plan on which type of human-robot communication is most effective and will be used throughout this project. In this case the human-robot communication choice was speech synthesis & recognition. The form of testing the tour-guide was an agile testing strategy.

The artefact development section discusses the assembling of the hardware and the individual components and purpose of each. Alongside the software of the autonomous movement feature and the human-robot communication. The software utilised ROS and developed each package using Python2 & Python3. Each package was combined into a main file so each one can be used together simultaneously.

The testing, results & evaluation section covered the experimentation for the speech synthesis & recognition and the autonomous movement packages. Each experiment further improved the tour-guide and made it more seamless for users to interact with the tour-guide. Each section was also evaluated and discussed further to determine whether goals were achieved.

In conclusion the main aim has been achieved and all four objectives were successfully completed. The tour-guide consists of a Turtlebot3 Waffle Pi chassis with a USB microphone & speaker and software that listens and speaks to people alongside autonomously moving to desired locations. The tour-guide is able to coherently describe each location and multiple names can be provided for one location.

Bibliography

[1] Amazon. Build skills with the alexa skills kit. https://developer.amazon.com/en-US/docs/alexa/ask-overviews/build-skills-with-the-alexa-skills-kit.html.

[2] Natesh M Bhat. pyttsx3 2.87. https://pypi.org/project/pyttsx3/.

[3] Alexander Briones. The different types of mics and their uses. https://www.gearank.com/articles/types-of-mics.

[4] ROBOTIS e Manual. Machine learning. http://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/#machine-learning.

[5] ROBOTIS e Manual. Slam. http://emanual.robotis.com/docs/en/platform/turtlebot3/ros2_slam/#cartographer.

[6] ROBOTIS e Manual. Specifications. http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/.

[7] Guru99. What is agile testing? process, strategy, test plan, life cycle example. https://www.guru99.com/agile-testing-a-beginner-s-guide.html.

[8] Faisal Hoque. Full demo tour of tour-guide from starting point. https://drive.google.com/file/d/1ln6BnFhAvJWrI3sW_7ZWDMZmhu0Pj6oU/view?usp=sharing.

[9] Faisal Hoque. Full tour of tour-guide in reverse. https://drive.google.com/file/d/1dZcLwniZ-7MIjyEbf0lmnxLH1k9BHWZ5/view?usp=sharing.

[10] Faisal Hoque. Move to small room. https://drive.google.com/file/d/1zgwbORLbx7PsrOs4ZBxwOuTVMpcbZMi1/view?usp=sharing.

[11] Faisal Hoque. Move to toilet. https://drive.google.com/file/d/1AM8i1VsXZqhl781kR8xd7ZQ1En4oVDv5/view?usp=sharing.

[12] Faisal Hoque. Testing the speech recognition feature. https://drive.google.com/file/d/1uc5RvNZpAMHckCKQNxCGQufpakOnvK6v/view?usp=sharing.

[13] Faisal Hoque. Testing the speech synthesis feature. https://drive.google.com/file/d/15EHDMPZ3JLd_PpjWR3Q9B7DFXSFaMzq4/view?usp=sharing.

[14] David V. Lu. move_base. http://wiki.ros.org/move_base.

[15] MathWorks. Matlab - mathworks. https://uk.mathworks.com/products/matlab.html.

[16] Raspberry Pi. Camera module v2. https://www.raspberrypi.org/products/camera-module-v2/.

[17] Raspberry Pi. Raspberry pi 3 model b+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[18] Python. json — json encoder and decoder. https://docs.python.org/3/library/json.html.

[19] PyTorch. Pytorch. https://pytorch.org/.

[20] ROBOTIS. Bt-410 set. http://www.robotis.us/bt-410-set/.

[21] ROBOTIS. Lds-01. http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/.

[22] ROBOTIS. Lipo battery 11.1v 1800mah lb-012. http://www.robotis.us/lipo-battery-11-1v-1800mah-lb-012/.

[23] ROBOTIS. Opencr1.0. http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_opencr1_0/.

[24] ROBOTIS. Robotis engineer kit 1 [us]. http://www.robotis.us/robotis-engineer-kit-1-us/.

[25] ROBOTIS. Robotis gp. http://www.robotis.us/robotis-gp/.

[26] ROBOTIS. Robotis premium. http://www.robotis.us/robotis-premium/.

[27] ROBOTIS. [ros 1] navigation. http://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/#ros-1-navigation.

[28] ROBOTIS. Specifications. http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/#specifications.

[29] ROBOTIS. Tb3 ball caster-a01 (1ea). http://www.robotis.us/tb3-ball-caster-a01-1ea/.

[30] ROBOTIS. Tb3 pcb support-ibb-01 (12ea). http://www.robotis.us/tb3-pcb-support-ibb-01-12ea/.

[31] ROBOTIS. Tb3 waffle plate-ipl-01 (8ea). http://www.robotis.us/tb3-waffle-plate-ipl-01-8ea/.

[32] ROBOTIS. Tb3 wheel tire set-isw-01 (2ea). http://www.robotis.us/tb3-wheel-tire-set-isw-01-2ea/.

[33] ROBOTIS. Xm430-w210 specifications. http://emanual.robotis.com/docs/en/dxl/x/xm430-w210/.

[34] Rothenburg Ron. Real estate robots. https://www.cs.cmu.edu/~minerva/press/realprogress/index.html, 1998.

[35] scitkit learn. scikit-learn: machine learning in python. https://scikit-learn.org/stable/.

[36] Wolfram Burgard Armin B. Cremers Frank Dellaert Dieter Fox Dirk Hahnel Charles Rosenberg Nicholas Roy Jamieson Schulte Dirk Schulz Sebastian Thrun, Maren Bennewitz. Minerva: A second-generation museum tour-guide robot. February 1999.

[37] Wolfram Burgard Armin B. Cremers Frank Dellaert Dieter Fox Dirk Hahnel Charles Rosenberg Nicholas Roy Jamieson Schulte Dirk Schulz Sebastian Thrun, Maren Bennewitz. Minerva: A tour-guide robot that learns. January 1999.

[38] Tensorflow. Tensorflow. https://www.tensorflow.org/.

[39] Theano. Welcome. http://deeplearning.net/software/theano/#.

[40] UKHONK. Ukhonk portable usb speaker with loud stereo sound,usb powered stereo speaker for computer,notebook,laptop,pc,home,outdoors,travel,checkout counter. https://www.amazon.co.uk/gp/product/B07K85H3V2/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&th=1.

[41] UNOOE. Conference microphone unooe microphone for computer with omnidirectional boundary condenser plug & play usb pc mic for video conference skype recording gaming. https://www.amazon.co.uk/gp/product/B083LKSGXW/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1.

[42] John Nicholson Sachin Pavithran Vladimir Kulyukin, Chaitanya Gharpure. Rfid in robot-assisted indoor navigation for the visually impaired. November 2004.

[43] Wit.ai. Natural language for developers. https://wit.ai/.

[44] Dieter Fox Dirk Hahnel Gerhard Lakemeyery Dirk Schulz Walter Steiner Wolfram Burgard, Armin B. Cremers and Sebastian Thrunz. The interactive museum tour-guide robot. January 1998.

[45] Anthony Zhang. Speechrecognition 3.8.1. `https://pypi.org/project/SpeechRecognition/`.

Appendices

Appendix A

Artefact Development

A.1 Hardware Components

A.1.1 Turtlebot3 Waffle Pi

Figure A.1: Raspberry Pi 3 Model B+

Figure A.2: LDS-01 side view

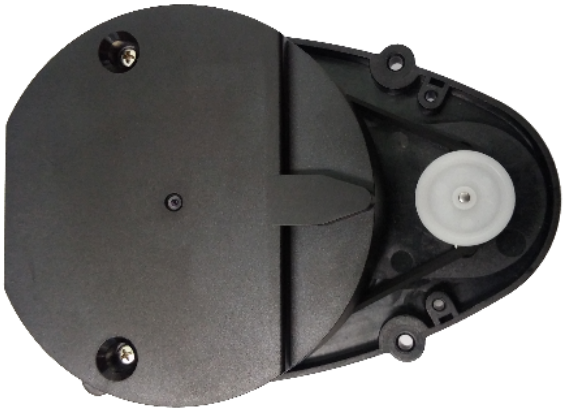


Figure A.3: LDS-01 top view

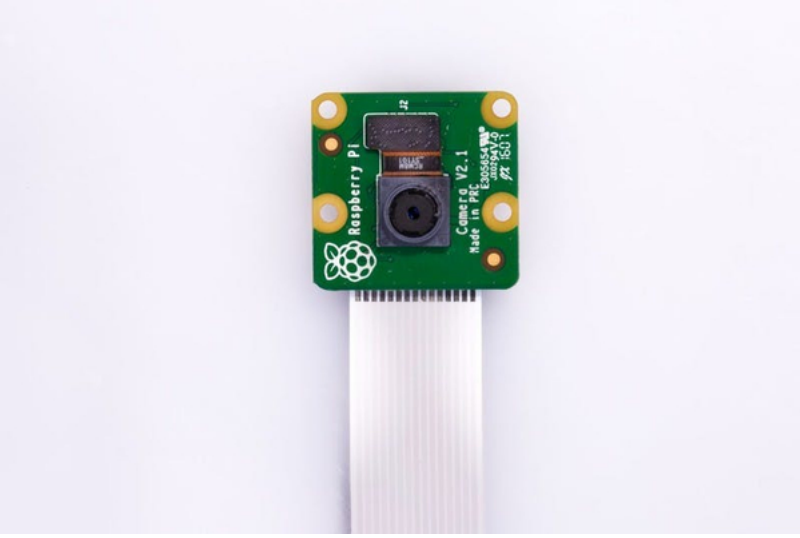


Figure A.4: Raspberry Pi Camera V2

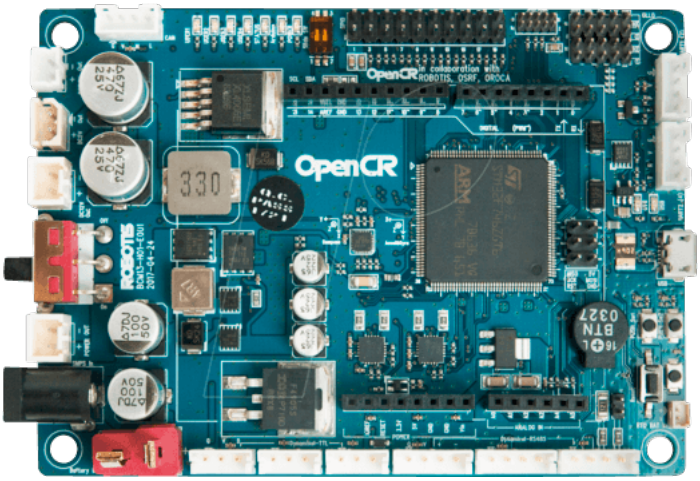


Figure A.5: OpenCR1.0 Board

Figure A.6: XM430 Dynamixel Motor

Figure A.7: BT-410 Set

Figure A.8: Li-Po Battery 11.1v 1800mAh LB-12

Figure A.9: TB3 Waffle Plate

Figure A.10: TB3 Wheel Tire Set

Figure A.11: TB3 Ball Caster

Figure A.12: TB3 PCB Support

A.1.2 Human-Robot Communication

Figure A.13: UKHONK Portable USB Speaker Model A [40]

A.1.3 Combining Components

Figure A.14: UKHONK Portable USB Sound Bar [40]

Figure A.15: Assembling Turtlebot3 Waffle Pi third layer

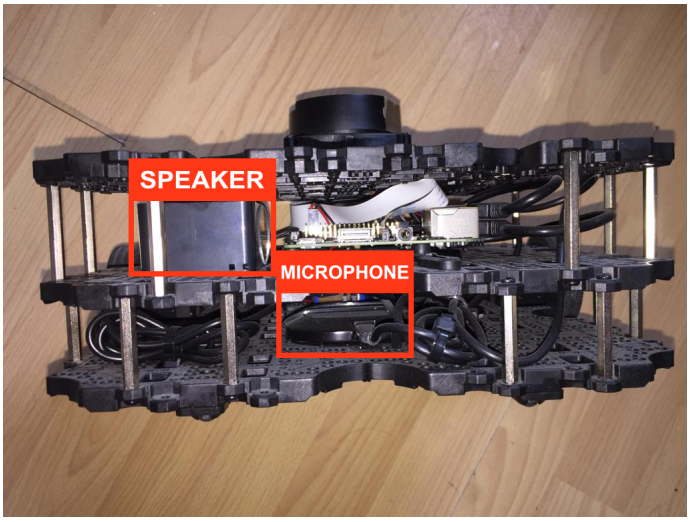


Figure A.16: Assembling the speaker and microphone on the Turtlebot3 Waffle Pi

A.2 Developing Software

A.2.1 Autonomous Movement

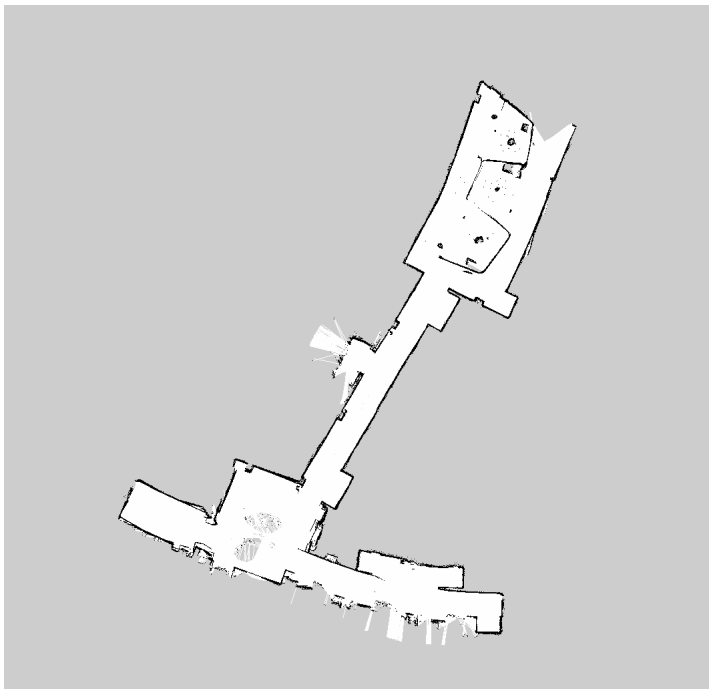


Figure A.17: SLAM Mapping of the STEM building

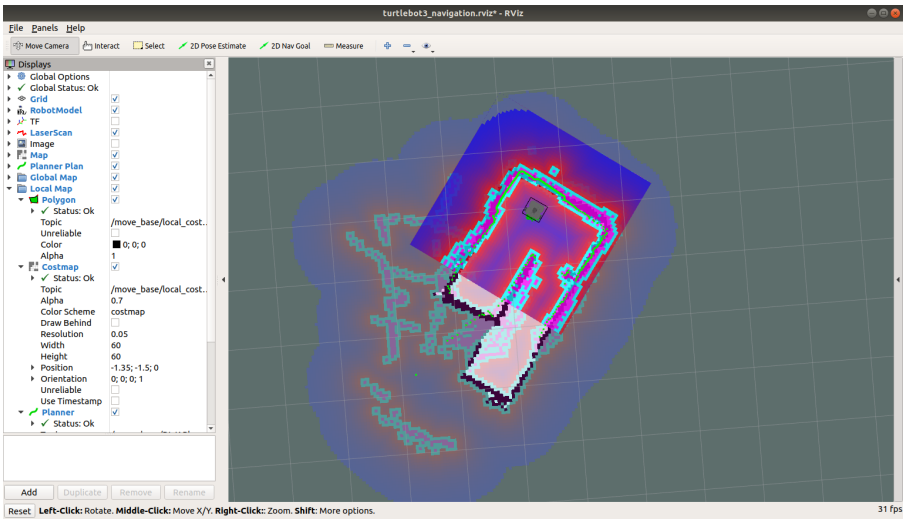


Figure A.18: Navigation node displaying the map with RViz, with correct pose

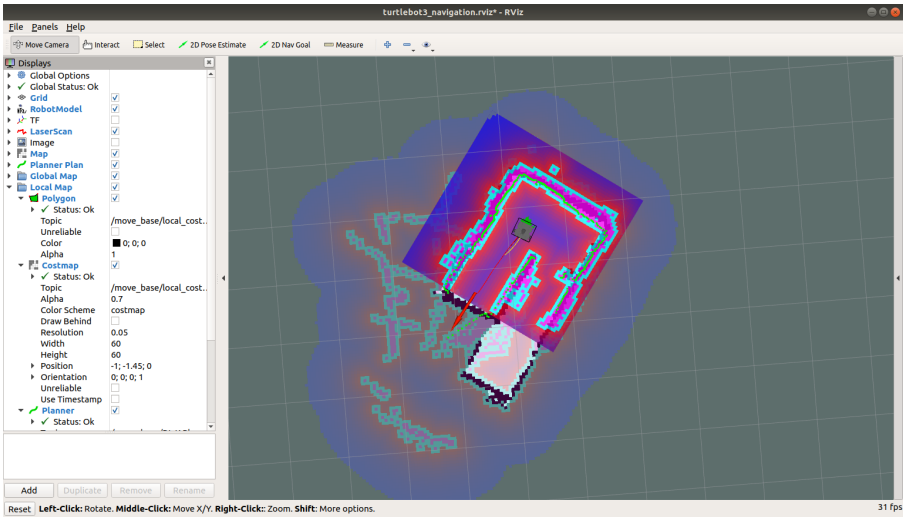


Figure A.19: Moving the tour-guide using 2D Nav Goal

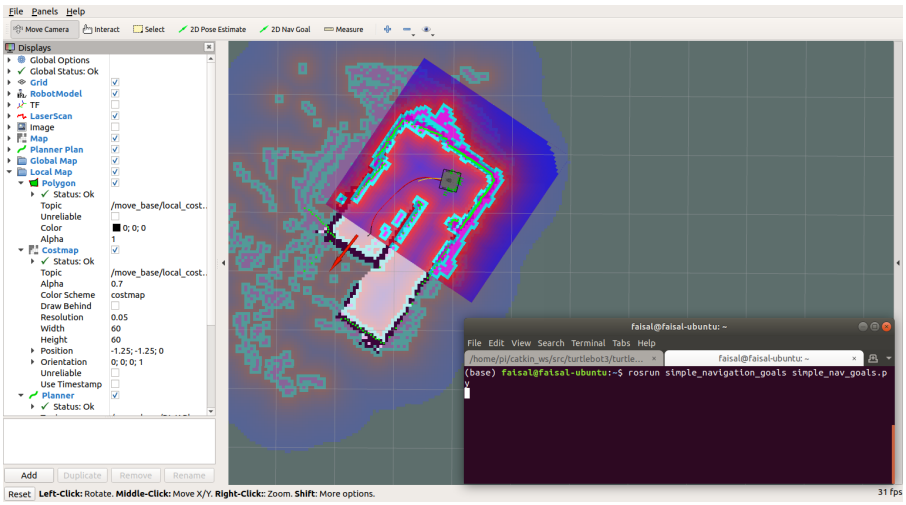


Figure A.20: Moving the tour-guide using custom Python code

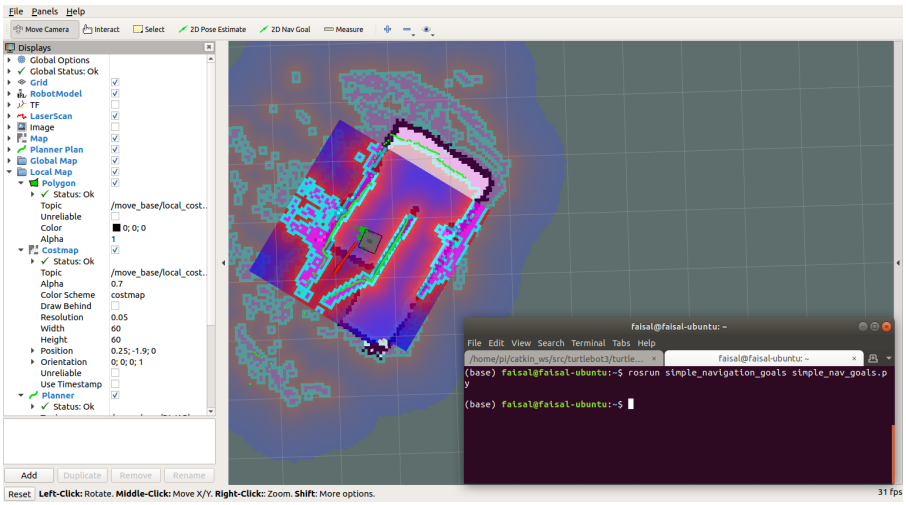


Figure A.21: Completed movement with Python code

A.2.2 Python Code

run_robot.py

```python
#!/usr/bin/env python3

from rospy import wait_for_service, ServiceProxy, ServiceException,
     Time, sleep, init_node, is_shutdown, Subscriber
from actionlib import SimpleActionClient
from uob_tour_guide.srv import Speak, Listen
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import PoseWithCovarianceStamped
from std_msgs.msg import String
from utility import get_map_nodes, get_tour_nodes
import json

class RunRobot:
    def __init__(self):
        print('Robot running...')
        self.amcl_pose = None
        self.current_location = None

        init_node('uob_tour_guide', anonymous=True)

        while not is_shutdown():
            self.listen_to_robot()

    # TODO: Implement "Ok Robot" before asking for tours?
    # TODO: Implement help feature that loops through available
    locations?
    def listen_to_robot(self):
        speech = self.robot_listen()
        print(speech)
```

```
28          map_nodes = get_map_nodes()
29
30          if speech['_text'] and speech['entities']:
31              if 'intent' in speech['entities'] and speech['entities'
    ]['intent'][0]['value'] == 'tour':
32                  self.robot_speak('Do you want a full tour around
    the building?')
33                  new_speech = self.robot_listen()
34                  print(new_speech)
35
36                  if 'intent' in new_speech['entities'] and
    new_speech['entities']['intent'][0]['value'] == 'true':
37                      self.robot_speak('Follow me for the tour')
38                      print('Tour method here')
39                      self.get_amcl_pose()
40                      tour_nodes = get_tour_nodes(map_nodes=map_nodes
    , location={ 'x': self.current_location.x, 'y': self.
    current_location.y })
41
42                      for node in tour_nodes:
43                          print(f'Moving to: {node["position"]}')
44                          self.robot_speak(f'Follow me to the {node["
    name"]}')
45                          self.robot_move(node)
46
47                          self.robot_speak(node['description'])
48                          sleep(0.5)
49
50                      self.robot_speak('That concludes the tour.')
51
52                  elif 'intent' in new_speech['entities'] and
    new_speech['entities']['intent'][0]['value'] == 'false':
53                      self.robot_speak('Cancelling full tour')
54                  else:
55                      self.robot_speak('Sorry, I didn\'t catch that,
    cancelling full tour')
56              elif 'move_to' in speech['entities']:
57                  for node in map_nodes:
58                      keyword = speech['entities']['move_to'][0]['
    value'].lower()
59                      keywords = [word.lower() for word in node['
    keywords']]
60
61                      if keyword == node['name'].lower() or keyword
    in keywords:
62                          self.robot_speak(f'Shall I move to {keyword
    }?')
63                          new_speech = self.robot_listen()
64                          print(new_speech)
65
66                          if 'intent' in new_speech['entities'] and \
67                              new_speech['entities']['intent'][0]['
    value'] == 'true':
68                              self.robot_speak(f'Moving towards {
    keyword}')
69                              print(f'Moving to: {node["position"]}')
70
```

```
                            self.robot_move(node)
                            self.robot_speak(node['description'])

                        elif 'intent' in new_speech['entities'] and \
                            new_speech['entities']['intent'][0]['value'] == 'false':
                            self.robot_speak(f'Cancelling movement to {keyword}')
                        else:
                            self.robot_speak(f'Sorry, I didn\'t catch that, cancelling movement to {keyword}')

    def robot_move(self, map_node):
        client = SimpleActionClient('move_base', MoveBaseAction)
        client.wait_for_server()

        goal = MoveBaseGoal()

        goal.target_pose.header.frame_id = map_node['frame_id']
        goal.target_pose.header.stamp = Time.now()

        goal.target_pose.pose.position.x = map_node['position']['x']
        goal.target_pose.pose.position.y = map_node['position']['y']
        goal.target_pose.pose.position.z = map_node['position']['z']

        goal.target_pose.pose.orientation.x = map_node['orientation']['x']
        goal.target_pose.pose.orientation.y = map_node['orientation']['y']
        goal.target_pose.pose.orientation.z = map_node['orientation']['z']
        goal.target_pose.pose.orientation.w = map_node['orientation']['w']

        client.send_goal(goal)
        client.wait_for_result()

        return client.get_result()

    def robot_speak(self, value):
        wait_for_service('robot_speak')

        try:
            speak = ServiceProxy('robot_speak', Speak)
            speak(value)
        except ServiceException as e:
            print(e)

    def robot_listen(self):
        wait_for_service('robot_listen')

        try:
            listen = ServiceProxy('robot_listen', Listen)
```

```
            response = listen()
            replace_chars = ['u\'', '\'']
            data = response.data
            for char in replace_chars:
                data = data.replace(char, '"')

            return json.loads(data)
        except (ServiceException, json.decoder.JSONDecodeError) as e:
            print(e)
            return {'_text': '', 'entities': {}}

    def assign_amcl_pose(self, result):
        self.amcl_pose = result
        self.current_location = result.pose.pose.position

    def get_amcl_pose(self):
        sub = Subscriber('amcl_pose', PoseWithCovarianceStamped, self.assign_amcl_pose)
        sleep(0.1)
        sub.unregister()


if __name__ == '__main__':
    RunRobot()
```

Listing A.1: Main Python code that runs the robot

__init__.py

```
#!/usr/bin/env python3

import json
import os
import math
script_dir = os.path.dirname(__file__)


def get_map_nodes(file_path=f'{script_dir}/map_nodes.json'):
    with open(file_path) as json_file:
        map_nodes = json.load(json_file)
    json_file.close()

    return map_nodes

def xy_dist(x1, y1, x2, y2):
    X2 = abs(max(x1, x2) - min(x1, x2))
    Y2 = abs(max(y1, y2) - min(y1, y2))
    return math.sqrt(X2**2 + Y2**2)

def get_tour_nodes(map_nodes, location, nodes_len=0, tour_nodes=None):
    if tour_nodes is None:
        tour_nodes = []
        map_nodes = [map_node for map_node in map_nodes if map_node['in_tour']]
    else:
        location = {
```

```
            'x': tour_nodes[len(tour_nodes) - 1]['position']['x'],
            'y': tour_nodes[len(tour_nodes) - 1]['position']['y']
        }

    if nodes_len <= 0:
        nodes_len = len(map_nodes)

    def sort_by_distance(item):
        return item['distance']

    for i, node in enumerate(map_nodes):
        dist = xy_dist(location['x'], location['y'], node['position
    ']['x'], node['position']['y'])
        map_nodes[i]['distance'] = dist

    map_nodes.sort(key=sort_by_distance, reverse=False)

    tour_nodes.append(map_nodes[0])
    map_nodes.pop(0)

    if len(tour_nodes) >= nodes_len:
        return tour_nodes
    else:
        return get_tour_nodes(map_nodes=map_nodes, location=
    location, nodes_len=nodes_len, tour_nodes=tour_nodes)
```

Listing A.2: Utility functions for run_robot.py

listen.py

```
#!/usr/bin/env python

import rospy
from robot_speech.srv import String, Listen
from speech_recognition import Recognizer, Microphone,
    UnknownValueError, RequestError


def robot_listen(request):
    try:
        recogniser = Recognizer()
        microphone = Microphone()

        print('Listening')

        with microphone as source:
            audio = recogniser.listen(source)

        return str(recogniser.recognize_wit(audio, key='2
    F35S7KRNBLUIIFEVKC2PLFQ2XPGA45L', show_all=True))
    except (UnknownValueError, RequestError) as e:
        print(e)
        print('\n')
        return str({'_text': '', 'entities': {}})

def run_server():
    rospy.init_node('robot_listen_server')
    service = rospy.Service('robot_listen', Listen, robot_listen)
```

```
    rospy.spin()


if __name__ == '__main__':
    run_server()
```

Listing A.3: ROS node that runs the listening code for the microphone

speak.py

```
#!/usr/bin/env python

import rospy
from robot_speech.srv import Speak
import pyttsx3


def handle_speak(request):
    text = request.data

    if text:
        engine = pyttsx3.init()

        for voice in engine.getProperty('voices'):
            if voice.name == 'english':
                use_voice = voice

        engine.setProperty('voice', use_voice.id)
        engine.say(text)
        engine.runAndWait()
        return True

    return False

def run_server():
    rospy.init_node('robot_speak_server')
    service = rospy.Service('robot_speak', Speak, handle_speak)
    print('Ready to speak')
    rospy.spin()


if __name__ == '__main__':
    run_server()
```

Listing A.4: ROS node that handles the speech for the tour-guide using the speaker

simple_nav_goals.py

```
#! /usr/bin/env python
import rospy
import actionlib

from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal


if __name__ == '__main__':
```

```
    rospy.init_node('simple_nav_goals')

    client = actionlib.SimpleActionClient('move_base',
    MoveBaseAction)
    client.wait_for_server()

    goal = MoveBaseGoal()

    goal.target_pose.header.frame_id = 'map'
    goal.target_pose.header.stamp = rospy.Time.now()

    goal.target_pose.pose.position.x = 0.615693867207
    goal.target_pose.pose.position.y = -0.547673225403
    goal.target_pose.pose.position.z = 0.0

    goal.target_pose.pose.orientation.x = 0.0
    goal.target_pose.pose.orientation.y = 0.0
    goal.target_pose.pose.orientation.z = 0.487061487393
    goal.target_pose.pose.orientation.w = 0.873367681735

    client.send_goal(goal)
    client.wait_for_result()

    print(client.get_result())
```

Listing A.5: First experiment code to make the tour-guide move using Python

map_nodes.py

```
[
  {
    "name": "Stairs",
    "description": "This is the staircase to the ground floor",
    "keywords": [
      "Staircase"
    ],
    "in_tour": false,
    "frame_id": "map",
    "position": {
      "x": 1.17293763161,
      "y": -1.87601912022,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.962862843537,
      "w": 0.269991008251
    }
  },
  {
    "name": "Toilet",
    "keywords": [
      "Bathroom",
      "Loo"
    ],
    "description": "This is the toilet on the first floor",
    "in_tour": false,
```

```
    "frame_id": "map",
    "position": {
      "x": 2.20678186417,
      "y": -1.60780787468,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": -0.291221218251,
      "w": 0.956655738519
    }
  },
  {
    "name": "Master Room",
    "description": "This is the master room of the house",
    "keywords": [],
    "in_tour": true,
    "frame_id": "map",
    "position": {
      "x": 2.31064152718,
      "y": -1.21090614796,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.468362223488,
      "w": 0.883536545712
    }
  },
  {
    "name": "Storage Room",
    "description": "Storage room for the first floor",
    "keywords": [],
    "in_tour": false,
    "frame_id": "map",
    "position": {
      "x": 1.63168299198,
      "y": -0.876165509224,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.473312744197,
      "w": 0.880894458026
    }
  },
  {
    "name": "Large Room",
    "description": "The is the second largest room on the first
    floor",
    "keywords": [],
    "in_tour": true,
    "frame_id": "map",
    "position": {
```

```
      "x": 0.615693867207,
      "y": -0.547673225403,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.487061487393,
      "w": 0.873367681735
    }
  },
  {
    "name": "Small Room",
    "description": "Smallest room on the first floor",
    "keywords": [],
    "in_tour": true,
    "frame_id": "map",
    "position": {
      "x": 0.501513361931,
      "y": -0.377310037613,
      "z": 0.0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.950513478964,
      "w": 0.310683321578
    }
  }
]
```

Listing A.6: JSON data file of all traversable locations on the home_corridor map

Appendix B

Testing, Results & Evaluation

B.1 Autonomous Movement

B.1.1 Full Tour

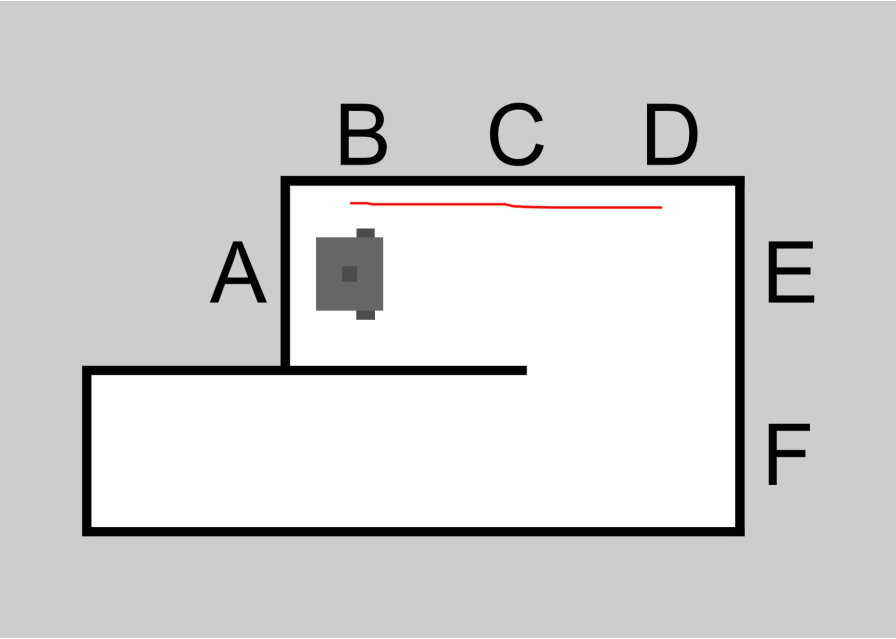


Figure B.1: Shows a diagram of the tour-guide moving from locations A to D

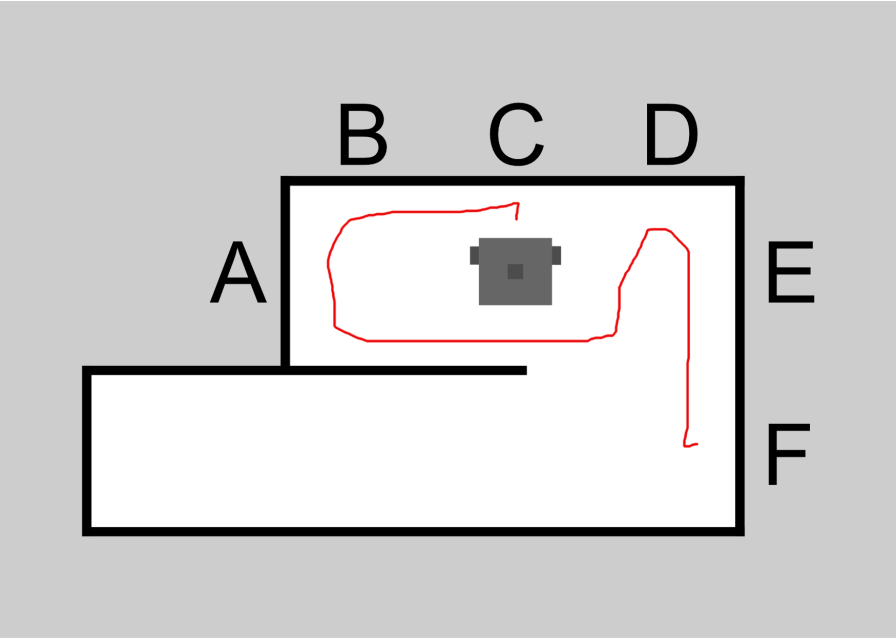


Figure B.2: Shows a diagram of the tour-guide starting at location C, then B, A, D, E and ending on F

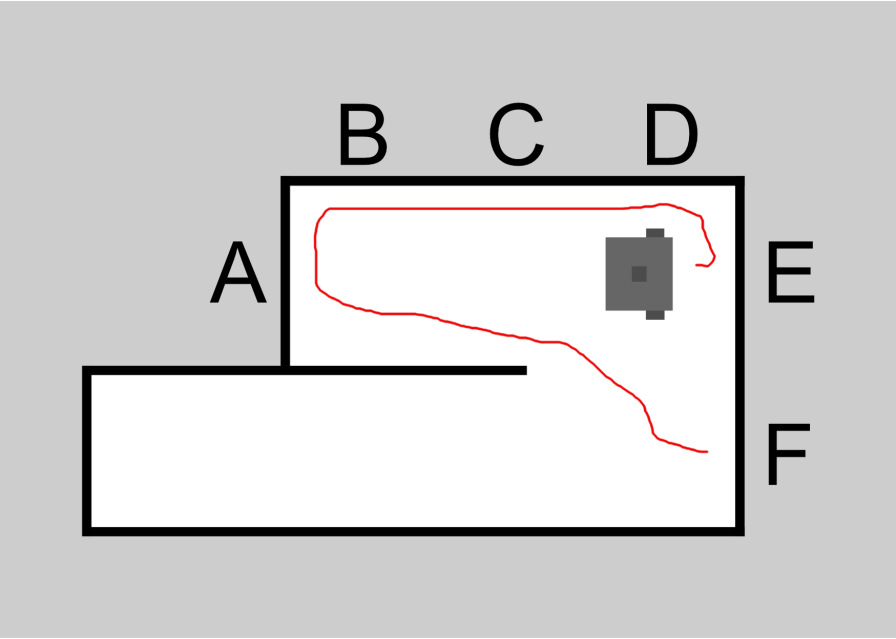


Figure B.3: Shows a diagram of the tour-guide starting at location E, then D, C, B, A and ending on F

Appendix C

Technical Documentation

C.1 Introduction

This documentation will show how to use the Tour-Guide software on a Turtlebot3 Waffle Pi and how to run each package. Once properly set-up the Turtlebot3 Waffle Pi should be able to autonomously navigate an area upon voice instruction once provided with a map and notable locations. This documentation assumes you have a basic understand of ROS and Turtlebot3, if not please refer to the before setup section for tutorials.

C.1.1 Before Installation

You will need to make sure you have properly installed ROS and all other Turtlebot3 packages as required for the Turtlebot3 Waffle Pi. Please follow all ROS and Turtlebot3 tutorials and instructions before attempting this, as it requires knowledge of ROS and Turtlebot3.

You can find the relevant documentation and the version of ROS and Turtlebot3 that this project followed below:

- ROS Tutorials - http://wiki.ros.org/ROS/Tutorials
- ROS Version - Melodic Morenia
- Turtlebot3 e-Manual - http://emanual.robotis.com/docs /en/platform/-turtlebot3/overview/
- Turtlebot3 Model - Waffle Pi
- Turtlebot3 Operating System - Raspbian
- Turtlebot3 ROS Version - Kinetic Kame

C.2 Installation

This installation assumes all relevant Turtelbot3 packages have been installed and the user has an intermediate understanding with ROS. The user should also make sure the Turtlebot3 Waffle Pi is connected to the same WiFi network as the host computer and knows how to SSH into the Waffle Pi, as shown in the Turtlebot3 documentation.

C.2.1 Turtlebot3 Waffle Pi Installation

Once all relevant packages have been installed from the Turtlebot3 website and the tutorials have been completed and understood. The robot speech package will need to be downloaded from GitHub using Git on the raspberry pi for the Turtlebot3.

Requirements

- GitHub Repo: https://github.com/FHaisal/robot_speech.git
- SpeechRecognition: https://pypi.org/project/SpeechRecognition/
- pyttsx3: https://pypi.org/project/pyttsx3/

Installation Steps

1. Once SSH'd run the following command: cd /catkin_ws/src && git clone https://github.com/FHaisal/robot_speech.git
2. Then run: pip install SpeechRecognition pyttsx3
3. Then run: cd /catkin_ws && catkin_make

You will then need to run: sudo nano /catkin_ws/src/robot _speech/src/listen.py and add your Wit.ai key in the wit_key variable.

C.2.2 Master Computer Installation

Assuming all relevant packages are installed for Turtlebot3 and ROS, you will then need to install the uob_tour_guide package from GitHub.

Requirements

- GitHub Repo: https://github.com/FHaisal/uob_tour_guide.git
- actionlib: http://wiki.ros.org/actionlib
- move_base_msgs: http://wiki.ros.org/move_base_msgs
- geometry_msgs: http://wiki.ros.org/geometry_msgs

The actionlib, move_base_msgs and geometry_msgs packages should already exist, however if not, then they can be installed the same way you install packages in catkin_ws/src.

Installation Steps

1. Open CMD and run: cd /catkin_ws/src && git clone https://github.com/FHaisal/uob_tour_guide.git
2. then run cd /catkin_ws && catkin_make

C.3 Setup

Before running the new packages installed, it is important to create a map using SLAM and the save map feature as shown in the Turtlebot3 documentation (http://emanual.robotis.com/docs/en/platform/turtlebot3/slam/). Once the map is saved you will then need to run the navigation node using the saved map as shown in the Turtlebot3 documentation.

C.3.1 Collecting Location Pose

In this section you will be shown how to retrieve the location pose of a tour location, for example if you want the toilets to be in the tour or a move to location, then you will need to retrieve the pose of the toilet in the map. You will then be able to create your own JSON formatted data for each location.

Steps To Get Pose

1. Master machine type: roscore
2. Turtlebot3 machine type: roslaunch turtlebot3_bringup turtlebot3_robot.launch
3. Master machine type: roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
4. Master machine type: roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml (Replace map.yaml with the name of your map)
5. Master machine type: rostopic echo /amcl_pose (As shown in figure C.3.1)

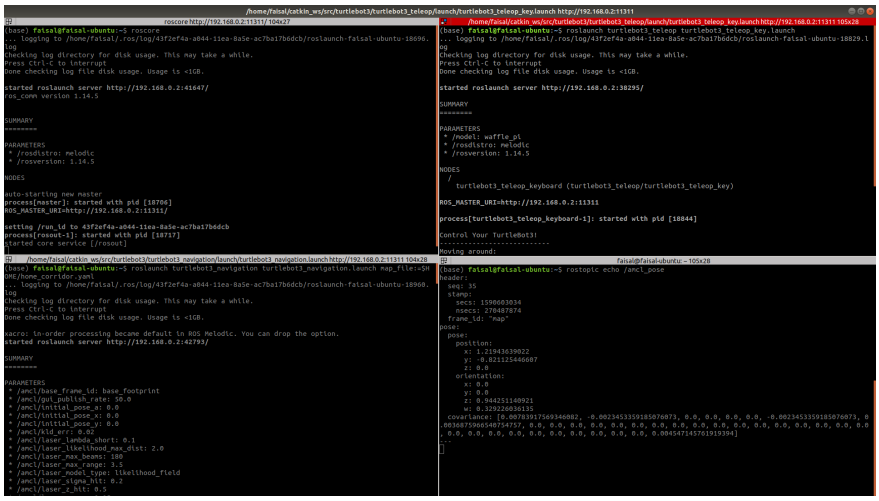


Figure C.1: Command line showing the commands ran to collect Turtlebot3 current pose

Creating The JSON File

Once the current pose of the Turtlebot3 Waffle Pi has been retrieved, you can now create the map_nodes.json file which is important for the Tour-Guide to know where the important locations are.

```
[
  {
    "name": "",
    "description": "",
    "keywords": [],
    "in_tour": false,
    "frame_id": "",
    "position": {
      "x": 0,
      "y": 0,
      "z": 0
    },
    "orientation": {
      "x": 0.0,
      "y": 0.0,
      "z": 0,
      "w": 0
    }
  }
]
```

Listing C.1: JSON file that contains the empty structure for the map_nodes.json file

As figure C.3.1 shows an array with a single object inside of it. It is fairly self explanatory on what each variable is, but they will need to be filled in using

the pose retrieved from the previous section, if multiple locations are required then just copy and paste the object as needed (it is important not to copy the array again).

- "name" - This should be the name of the location, for example toilet, this will serve as the primary keyword and name.
- "description" - This is a description that the tour-guide will speak to discuss the location.
- "keywords" - This is an array and you can put any other identifiers in there for the tour-guide to pick up as keywords. For example for the toilet, you can also add bathroom, loo or rest stop, etc.
- "in_tour" - This is a boolean value which will indicate whether you want this location to be included in the full tour. If you want it in the full tour then set this value to: true.
- "frame_id" - This is the id of the map, you can pull this information from the current pose information as shown above.
- "x" (position) - The x coordinate of the position node, can be pulled from the current pose as shown above.
- "y" (position) - The y coordinate of the position node, can be pulled from the current pose as shown above.
- "z" (position) - The z coordinate of the position node, can be pulled from the current pose as shown above.
- "x" (orientation) - The x coordinate of the orientation node, can be pulled from the current pose as shown above.
- "y" (orientation) - The y coordinate of the orientation node, can be pulled from the current pose as shown above.
- "z" (orientation) - The z coordinate of the orientation node, can be pulled from the current pose as shown above.
- "w" (orientation) - The w coordinate of the orientation node, can be pulled from the current pose as shown above.

It is important to locate the map_nodes.json file which will be located in /catkin_ws/src/uob_tour_guide/src/utility and replace the old map_nodes.json file with your new one. You can also add new objects to that one or just replace the code with your one.

C.4 Running Tour-Guide

Assuming the steps above were followed and there are no errors then running the tour-guide is quite simple by just running a few extra packages, as shown in the following steps.

1. Master machine: roscore
2. Turtlebot3 machine: roslaunch turtlebot3_bringup turtlebot3_robot .launch
3. Turtlebot3 machine: rosrun robot_speech listen.py
4. Turtlebot3 machine: rosrun robot_speech speak.py
5. Master machine: roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
6. Master machine: roslaunch turtlebot3_navigation turtlebot3_navigation .launch
7. Master machine: rosrun uob_tour_guide run_robot.py

Autonomous Tour Guide with Open-Source Software and the Internet of Things

BSc (Hons) Artificial Intelligence and Robotics (with Professional Practice Year)

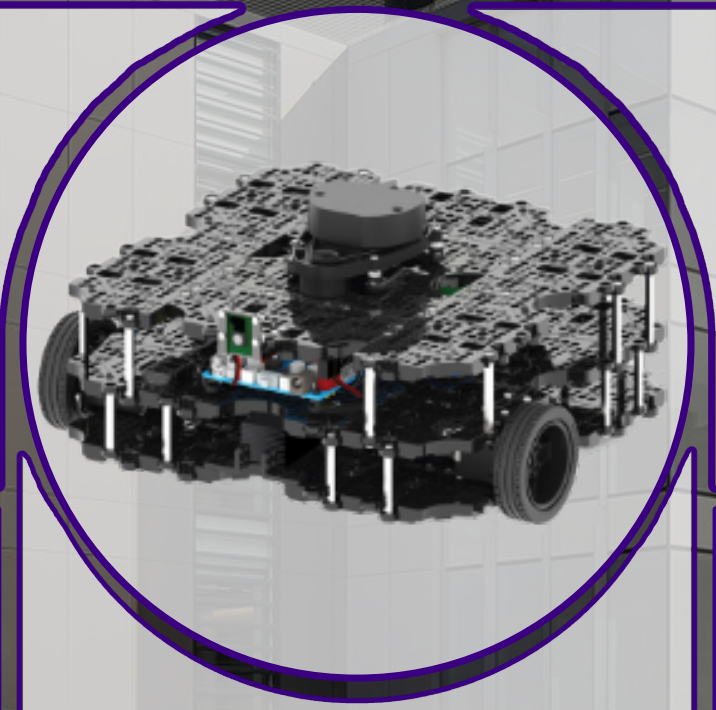
Student: Faisal Hoque **Student ID: 1618196** **Supervisor: Dayou Li**

Project Aim

This research aims to develop an autonomous tour guide robot with readily available hardware and open-source software at a fraction of the costs of previous robots with similar functionalities.

Objectives

- Review literature on autonomous tour guides
- Construct turtlebot3 waffle pi chassis
- Develop software to move autonomously
- Develop voice recognition & speech synthesis



Testing & Results

The main test is for the autonomous movement to see if the tour-guide reacts to speech and gives a full tour or moves to a specified location. For example: "show me around", will trigger a full tour as shown in image A. Whereas: "show me the toilet" will move towards the toilet as shown in image B

Image A

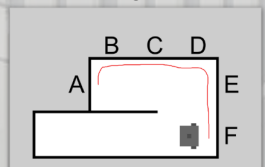
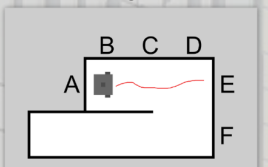


Image B



Artefact Development

The first step was to assemble the Turtlebot3 Waffle Pi components to enable movement, alongside the extra microphone and speaker for the voice recognition and speech synthesis

The second step was to develop the package that enabled autonomous movement using navigation and path planning. Then to also develop the package that enables speech synthesis and voice recognition using python

Technologies

ROS, TURTLEBOT3, python, Wit.AI

Conclusion

- Main aim has been achieved with all objectives
- Assembled chassis with Turtlebot3 Waffle Pi kit
- USB microphone and speaker attached
- Developed speech synthesis & voice recognition
- Developed autonomous movement with python in ROS
- Features were seamlessly integrated

List of Figures

List of Tables

List of Acronyms

ROS Robot Operating System

DQN Deep Q-Network

GUI Guided User Interface

SLAM Simultaneous Localization and Mapping

TB3 Turtlebot3

LIPO Lithium-Ion Polymer

LDS Laser Distance Sensor

LIDAR Light Detection and Ranging

BT Bluetooth

JSON JavaScript Object Notation

CMD Command Prompt

SSH Secure Socket Shell

WiFi Wireless Fidelity

STEM Science, Technology, Engineering and Mathematics

USB Universal Serial Bus

PCB Printed Circuit Board

COVID-19 Corona Virus Disease 2019

API Application Programming Interface