ASHESI

ASHESI UNIVERSITY

The Turtlebot Tour Guide (TTG)

APPLIED PROJECT

B.Sc. Management Information System

Ethel Adongo

2019

THE TURTLEBOT TOUR GUIDE

APPLIED PROJECT

Applied project submitted to the department of Computer Science and Information Systems (CSIS), Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Management Information Systems.

ETHEL ADONGO

2019

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……………………………………………………………………………………………

Candidate's Name:

……………………………………………………………………………………………

Date:

……………………………………………………………………………………………

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied projects laid down by Ashesi University.

Supervisor's Signature:

……………………………………………………………………………………………

Supervisor's Name:

……………………………………………………………………………………………

Date:

……………………………………………………………………………………………

Acknowledgements

Special thanks to God, my supervisor (who was an able and available guide for the duration of the project and beyond) and colleagues (who helped me with this project by sharing in my enthusiasm for the project and appreciating all the minor and major successes at every point).

Abstract

The Turtlebot Tour Guide project is a project that was started in a robotics class. Since then, the project has been worked on within classes, by work-study students and even as capstone projects. The aim of the project is to model a Turtlebot to be a tour-guide that can effectively conduct tour processes within Ashesi University. The project involves mapping of Ashesi University, the creation of path-planning algorithms and other modules. This project plans to improve and complete previous work that has been done on the Turtlebot Tour Guide in a way that makes it functional and effective for tours within Ashesi University.

Table of Contents

Chapter 1: Introduction

1.1: Project Background and Aim

The Turtlebot Tour Guide project (TTG) was started in Ashesi University, a couple of years ago, in a robotics class. The aim of the project was to use a Turtlebot – which is a simple and open-source robot kit – to create a tour-guide that could effectively take visitors of Ashesi University on a tour of the school. Although it started as a course project, it soon became a project that the Computer Science and Information Systems (CSIS) department of Ashesi University took a special interest in.

The TTG was started for a number of reasons. The first was to provide a new touring experience to visitors of Ashesi University. The purpose was to give people a tour experience that went beyond what they were normally used to – which was a tour with human tour-guides. Using a Turtlebot - which is a simple open-source robot toolkit - was a step beyond the norm and gave people who visited Ashesi University more to look forward to during the tours.

Another reason for starting the TTG was to help the student ambassadors of Ashesi University in their work. The student ambassadors of Ashesi University are students and sometimes faculty members who take it upon themselves to take visitors of Ashesi University on a tour of the school. Because the student ambassadors are themselves students with tight schedules, they are sometimes time-constrained and are adversely affected by having to make time to take visitors of the university on a tour. The TTG was started to help the student ambassadors in such situations as well: as an able robotic tour-guide, the TTG can safely run tours on behalf of the student ambassadors.

Again, the TTG provides a setting that serves as a great way to study the use of robots in an African context. Very little research has been done into how people receive technological advancements, especially ones that they might believe threatens to replace their position in the working world – in the African context. The existence of the TTG can also help assess how receptive people in the African context are to technological advancements. Hopefully, the TTG project can inspire conversations around adopting technological advancements and the general attitude and fears towards allowing technology into our daily lives. In a best-case scenario, the existence of the TTG will inspire people, both users and creators of technology tools, to grow with the unending advancements being made in the field of technology and create a desire within people to use and create solutions with technology to the everyday problems in our part of the world.

1.2: Project History and Past Developments

This section outlines the work that has already been done on the TTG previously

1.2.1: Navigation: A lot has been done in the area of navigation of the Turtlebot. Modules and scripts exist to ensure that the robot can move from a given point to another successfully without bumping into objects like walls and pillars. The navigation module of this project was implemented using the Turtlebot's built in autonomous navigation system. Scripts were also written to co-ordinate the process. However, navigation in a dynamic environment has not been catered for in the previous work that has already been done – this means that, during a tour, the robot might easily bump into objects like chairs and people that it may not have initially recognized as a part of the path it was meant to traverse. A solution to this navigation problem could be to implement a pathplanning algorithm that caters for dynamic environments – like the D*lite path-planning algorithm which “combines aspects of A* search, the classic AI heuristic search method, and incremental search to plan near-optimal paths in partially known environments” [1]

1.2.2: Path-Planning Algorithms: The current work that has been done on the TTG has a path planner that finds the shortest path (path with the least cost) between two points and returns that path as the optimal route. The path-planning algorithm is based on Dijkstra’s algorithm and works by iteratively calculating for the shortest path between two interim and/or end points whilst ignoring already visited states till it gets to the destination state, then returns the sequence of paths visited as the optimal path. What this algorithm does is determine the shortest way to get from one point to another. It does not make allowances for custom tours based on time and area (landmark points).

1.2.3: Tour-Control: Current work on the project does not have a simplified tour setup or control system. As it stands, the TTG has to be started up by first bringing-up and running scripts on the Turtlebot and the workstation before any communication can be made via the user interface available. The problem with this is that the scripts and bringup steps involve long and tedious processes that have to be manually launched. These different processes are run on up to 9 terminals just to start the tour process. Because the TTG should work with or without the presence of someone with the technical knowhow needed to operate the tour system, it is hindering and less than optimal for so much technical processes to be required in the start-up process.

1.2.4: Mapping: Maps exist for the purpose of the TTG. However, these maps exist for a single block of the University. The consistent expansion of the University requires that more maps be generated to make the touring process wholesome. Aside from that, some of the maps created from previous work have very poor quality which makes it difficult for the robot to navigate through the map

1.3: Contributions

This current TTG project aims at improving upon previous work and making the features that were created in the previous system fully functional so that the TTG can give tours to visitors of Ashesi University. The contributions to the TTG project that are made in this project are based on the work that has already been done. The contributions are in three main areas and are as discussed below:

1.3.1: Custom Tours: The previous work only allowed default tour paths between any two points. One objective of this project is to give users or tourists the choice of a custom tour. The custom tours are going to be created based on how important the different parts of Ashesi University are, in the context of a tour for visitors. Simply, the algorithm will plan a path that gives a tour of the best parts of Ashesi University within a given time range. This will ensure that the visitor is able to see the best part of the school within the little time they are able to partake in the tour.

1.3.2: Mapping: The maps existing for the TTG are very limited and cover about 10% of the entire tour space. Another objective of this project will be to create maps of the different unmapped areas in the institution and feed that data into the system do that tours can be conducted over those spaces as well. In addition to that, the low-quality maps that were generated from previous work will be edited to improve their quality for the purpose of a smooth and effective tour experience.

1.3.3: Start-up Simplification: To minimize the technicalities involved in starting up the TTG and to make the start-up process of the TTG faster, this project will include the creation of an executable(s) that will run the necessary start-up commands, run the necessary scripts and initialize the needed communication protocols. This way, just about anybody with authorization can start the tour process without any difficulties.

In summary, these are a list of the objectives of the TTG project:

- Implementation of a priority-based path-planning algorithm
- Create module to automate start-up ○ Bring-up of Turtlebot and workstation (this is the process that awakens the Turtlebot or starts it and all other necessary scripts and packages so that the TTG can work) ○ Sourcing (this process involves setting the environment for the TTG by running the necessary files) ○ Script-running
- Mapping – this involves creating maps for the new areas in Ashesi University and editing the already existing ones so that they are of better quality

1.4: Related Work

In 1997, the robot 'RHINO' was deployed for 6 days in a heavily-populated museum [2]. RHINO was a robot built to be a tour-guide and the results of its testing demonstrated that it was a reliable replacement for human tour-guides. With the success of this revolutionary automation, the museum's attendance increased by more than 50%. This project provides a lot of information on how some of the key processes of the system were implemented. As in the TTG, these key processes include localization, collision avoidance and mapping. Again, the RHINO project serves as a reference for building an intuitive and interactive user interface for the robot. Because both the TTG and RHINO projects have the same basic elements, the architecture and design of the RHINO project will be referenced when assessing that of the TTG project. This will create a good benchmark since the RHINO project has been very successful.

Whilst the RHINO project provides practical and real-life information on automation and how to implement it in the tour-guide industry, Feng and Liu's project on autonomous robot navigation explores the design and concept of autonomous navigation systems from a more theoretical and philosophical perspective [3]. Their work is majorly theoretical and analyzes the topic with an emphasis on its application for vehicles (as in the case of self-driving cars). The project produces a structure for autonomous robot navigation that can be easily replicated on other systems. In addition, the project worked on a navigation algorithm that was built on the philosophy of professor Dickman. This algorithm will be used to help direct the creation the priority-based path planning algorithm in the TTG project. This project also provides information that can and will be used for more general robot features like sensing, control and even calibration.

Similar to Feng and Liu's work on autonomous navigation, Li et al also worked on a project that analyzed the process and programs it takes to make a robot navigate autonomously in an environment that has dynamic elements. [4]

The results of this project were detailed in a report titled *'Active localization with dynamic obstacles.'* Dynamic elements refer to objects like doors and people whose position in any environment changes from time to time. These kinds of environments are abundant in public spaces, like university campuses. It is important to understand robot navigation in dynamic environments since very little of the movement of objects in such an environment can be predicted - this might cause the robot to collide with obstacles and get lost in an area. The project demonstrates and develops methods and systems used to ensure that the robot can actively localize (know its position) itself in such an environment at any given point in time. Because the TTG will be deployed in a university campus as well, the data provided in this project on dynamic environments can be used as a guide.

Another relevant project to the TTG is one that was undertaken as a part of undergraduate research in the York College of Pennsylvania. [5] This project explores mapping an area using ROS extensively and even provides a program that can be used to convert floorplans of any area into a map that ROS can recognize and work with. Aside from using the program that was developed in the project (as will be seen later in the report), the TTG project will also draw from the mapping and scaling suggestions from the project to create good and workable maps of Ashesi University.

These projects are informative and relevant in all aspects of the implementation of the TTG – in matters that deal with the physical robot, the algorithms running on the robot and the concepts of navigation, mapping and localization.

1.5: Tools and Hardware

The basic framework of the entire Turtlebot Tour Guide (TTG) from previous work is shown in figure 1 below. The same structure will be maintained and used for this TTG project as well.

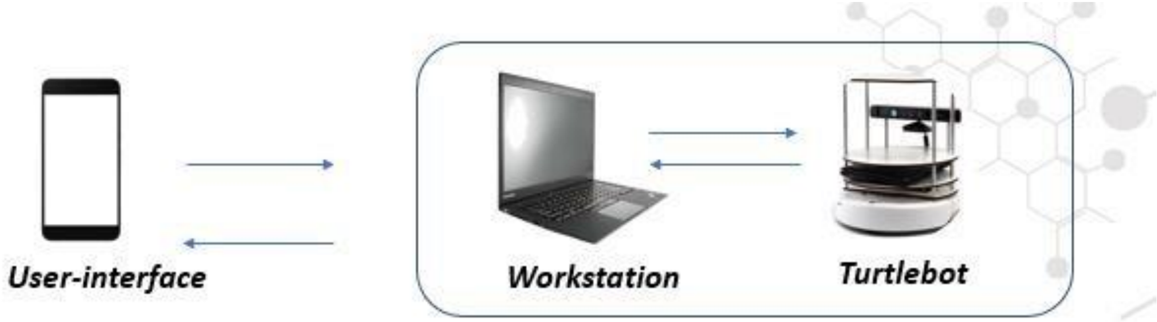


***Figure 1**. TTG hardware structure*

This TTG project will only work on the infrastructural requirements - the part that deals with the interaction and functionalities of the Turtlebot and its communication with the workstation. The workstation is an additional laptop that is not physically connected to the Turtlebot. It runs the Ubuntu operating system and is used to remotely control the Turtlebot. It also stores most of the scripts that are run to ensure that the TTG works as planned.

The Turtlebot, as mentioned earlier, is a simplified open-source robot tool kit that comes with a mobile base, a Kinect 3D sensor and a laptop that serves as its brain. The Turtlebot helps you "build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications" [6].

Figure 2 below is of its hardware structure. Even though the image below does not include a laptop, the Turtlebot comes with an Ubuntu Asus laptop connected to the Turtlebot base, and it serves as the processing unit of the entire system.

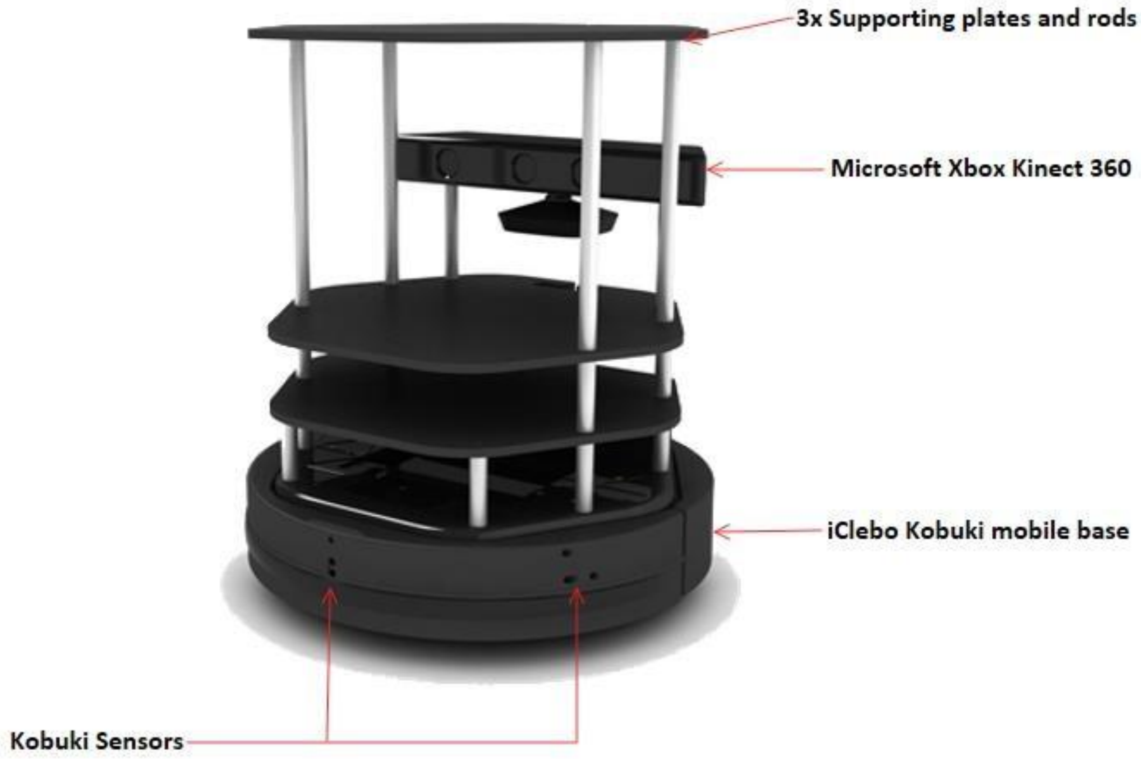

3x Supporting plates and rods
Microsoft Xbox Kinect 360
iClebo Kobuki mobile base
Kobuki Sensors

***Figure 2**. Labeled Turtlebot [6]*

The Turtlebot has wheels and so cannot move-up stairs. This hardware limitation requires that the paths it traverses have no stairs and are relatively smooth. Luckily, the campus of Ashesi University has been designed with accessibility in mind - ensuring that there is a ramp for almost every location that with stairs. This ensures that the Turtlebot is able to navigate to all the parts of Ashesi University that are relevant during a tour.

Other tools, technologies and resources used in this project will be discussed in chapter 3 of this report.

Chapter 2: Requirements Gathering and Analysis

2.1: Approach for gathering requirements

To ensure that the project was relevant to the people it was created for and the function it was meant to serve, requirements were gathered before the project was officially implemented. In this process, the aim was to find out what the end-users or stakeholders wanted to see in terms of functionality and features. The different stakeholders identified are stated below:

2.1.1: Tourists: These are the people who visit the host institution for work or recreational purposes and decide to take a tour to explore the scenery of the university. These people include students from other institutions, relatives and parents of students of the host institution, new employees and even new students.

2.1.2: Student ambassadors: Ashesi University currently has a tour-guide committee that takes charge of all the official touring activities in the school. The students or staff who volunteer to take visitors around the university are known as the (student) ambassadors. The TTG project draws a lot from the work of these ambassadors since it ultimately replicates their activities.

2.1.3: Computer Science and Information System (CSIS) Department: The TTG project was officially started by the CSIS department of Ashesi University. As the initiators of the project, their views on the TTG's functionalities and features are significant to the successful completion of the project. The CSIS department is also the official sponsor of the project – providing the tools and support needed to successfully complete the TTG project.

Aside from speaking to the stakeholders involved in the project, requirements were also gathered based on the previous work that has been done on the project. Previous reports on gathered requirements were incorporated to the new requirements identified to ensure that the TTG is fully functional. In addition, monitoring and observational activities were carried out to gather requirements from the tour experience in its most natural context.

2.2: Requirement Summary

Below are the various requirements gathered from the process stated above. These requirements also draw from the requirements that were gathered and completed in previous work:

2.2.1: Functional requirements

Time-Based custom tours – The TTG should give users or tourists the option of providing time-constraints. The TTG can then run computations to discover the best(shortest) route that covers the best part of the school within the time specified.

Extensive Maps – Users should be able to take tours of the newer places on campus, from hostel areas, staff offices through to class blocks and other support structures like the health center and cafeteria.

Easy start-up – It should be easy to start the TTG for a tour. The processes needed to start a tour with the TTG should not require too much technicalities.

2.2.2: Non-functional requirements

Performance – Because the TTG will be taking visitors on a tour, it is important that is able to execute the computational processes to plan a route fast enough to accommodate timepressed users. The TTG should also be able to communicate with its parts quickly to ensure that processes of the tour are undertaken in the fastest time possible – this requires that the network should be stable.

Scalability – The TTG project needs to be built to be scalable. With code and scripts that can be replicated, there could be multiple TTGs to facilitate all the tours that are conducted in Ashesi University. Also, the TTG should allow the storage of additional information that might be developed later in its use. For example, since the host institution is under constant expansion, the TTG should be able to take-in maps of newly developed regions without disturbing the tour experience.

Recoverability – The TTG needs to have a module for recovery that will ensure that unexpected and hindering events that halt the tour experience can be handled gracefully.

Accuracy – The maps generated, and the paths developed by the TTG should be accurate enough so that users are not led into restricted or dangerous areas. This requires that the TTG can navigate autonomously with a high level of accuracy. The accuracy will also have to be in the maps input and localization process of the TTG.

Security – The TTG needs to be secured against all types of attacks. For example, the TTG should be secured against theft – given that it is very portable. On the other hand, the TTG should also be secured against people who might illegally access the network and send commands that might override the default program. This could be hinder the TTG’s program and cause it to malfunction.

Operations – the Turtlebot Tour-Guide should be able to operate for the amount of time it takes to conduct a full-tour, in the worst-case scenario context. It should be able to support extensive use within that period of time and not crash.

The diagram below (figure 3) is a use-case diagram for a typical user of the TTG.

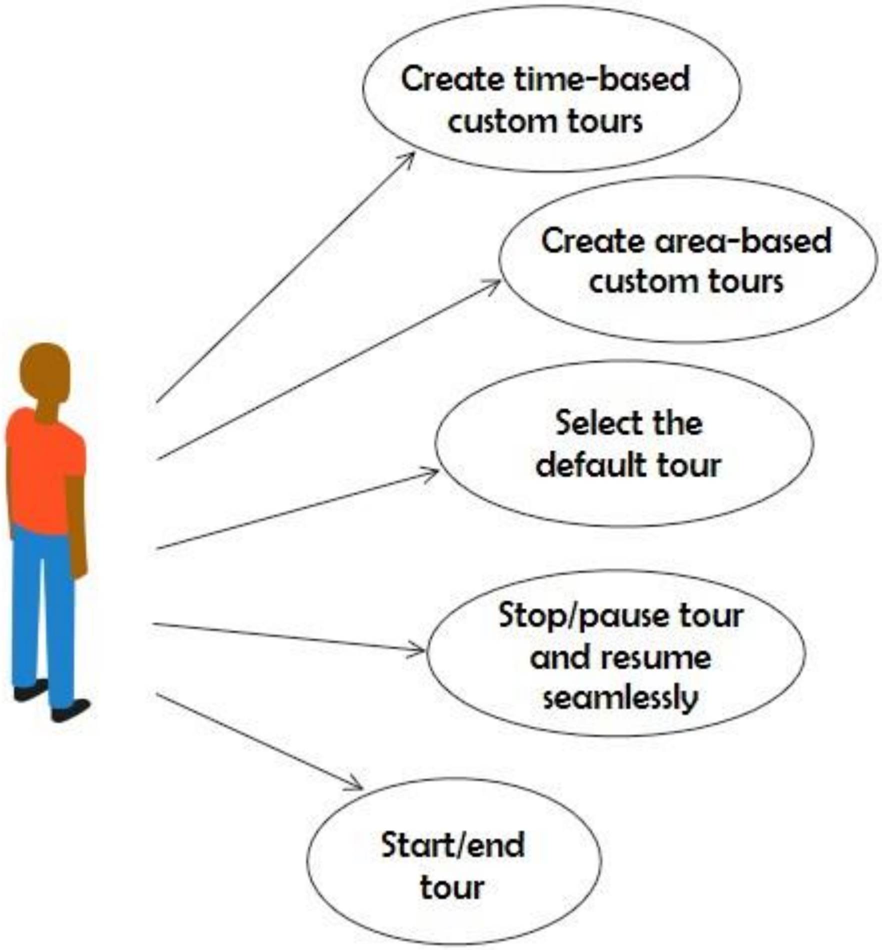

Create time-based custom tours
Create area-based custom tours
Select the default tour
Stop/pause tour and resume seamlessly
Start/end tour

Figure 3. *Use-case diagram*

The development process for the TTG is agile and so, requirements were gathered iteratively throughout the process. User tests and prototype methods like the Wizard of Oz were conducted periodically to ensure that the functionality of the TTG was meeting the requirements of the different stakeholders.

2.3: Priority requirements for path-planning algorithm

As stated early on, this project aims at developing a path-planning algorithm that will develop a route for a user within a given time, using rankings of the different points in Ashesi University. This requires that rankings or ratings of different places within Ashesi University be developed. To get this information, an anonymous survey was conducted. This survey asked for information on the tour-guide experience of the participant and then went on to ask for the user to rate a list of places given from 1 to 5, where 1 meant that the area or point in the university was not important during a tour and 5 meant that that area needed to be seen during a tour.

The survey was sent out to students of Ashesi University. Tourists or prospective tourists were not targeted during the survey because it was difficult to identify them and get in touch with them. Again, because most of these tourists come into the university without knowing what to expect, it would be difficult for them to decide on what they think is more important to see in the school. Hence, the survey targeted students of Ashesi University, all of who know the areas and some of who have given tours to visitors themselves. The full questionnaire is provided in appendix A.1.

After the questions were sent out, 17 responses were received and below is a summary of the statistics of the survey:

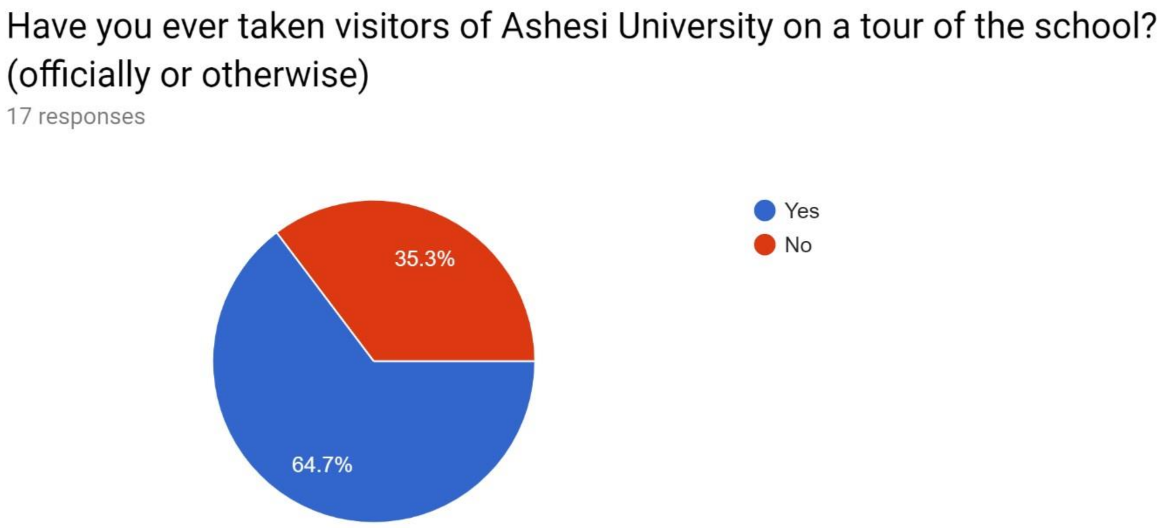


Fig 4. *Respondents who have given tours before*

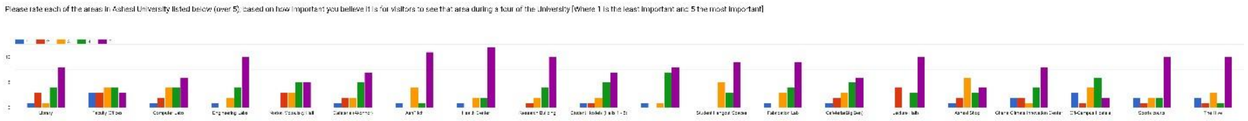


Fig 5. *Respondents rating of the various areas in Ashesi University*

A spreadsheet version of the responses for the rating (figure 5) is in appendix A.2.

About 65% of the respondents of the survey had some experience giving tours of Ashesi University to visitors. This shows that majority of the responses on the rankings for the different areas in Ashesi University are from people who have a good chance of knowing what visitors of Ashesi University enjoy seeing – which makes the data more accurate.

To get the ranking for each of the areas in the list, the average of the scores assigned were taken and rounded to a whole number; the results of that were then used as the rating or ranking of the area. Below are the different areas and their average ratings that were used for the priority-based path planning algorithm:

Table 1: *Ratings for places in Ashesi University*

Area	**Rating(average)**
Library	4
Faculty Offices	3
Computer labs	4
Engineering labs	4
Norton-Motoulsky hall	4
Cafeteria (Akonnor)	4
Ashesi Football Pitch	5
Health Center	4
Research Building	4
Student Hostels (Halls 1-8)	4
Student Hostels (Halls A-F)	4
Student Hangout Spaces	4
Fabrication Lab	4
Cafeteria (Big Ben)	4
Lecture Halls	4
Ashesi Shop	3
Ghana Climate Innovation Center	4
Off-Campus Hostels	3
Sports Courts	4
The Hive	4

Chapter 3: Architecture and Design

This section discusses the technology, frameworks, design and architecture of the Turtlebot Tour Guide Project.

3.1: General-Design:

The TTG utilizes both a Lenovo laptop and an Asus Laptop (that works with the Kobuki robot-base) for the project. Both the detached workstation (Lenovo Laptop) are registered on a local network to enable communication. The Turtlebot contains modules that manage which maps to load and which sounds to play for any tour initialized whilst the workstation contains modules that deal with moving the Turtlebot, planning paths, controlling the tour and interacting with the User Interface. The workstation deals with most of the computations needed for the TTG to run. The modules and how they relate with each other will be discussed in the section below.

3.2: Modules:

The modules and framework for the project was developed in earlier work and include a path-planning module, a movement module, a tour-control module, a mobile-relay module, a loader module (for maps), and a sound module. A diagram showing the flow of data through these modules is as shown in the figure below. [11]

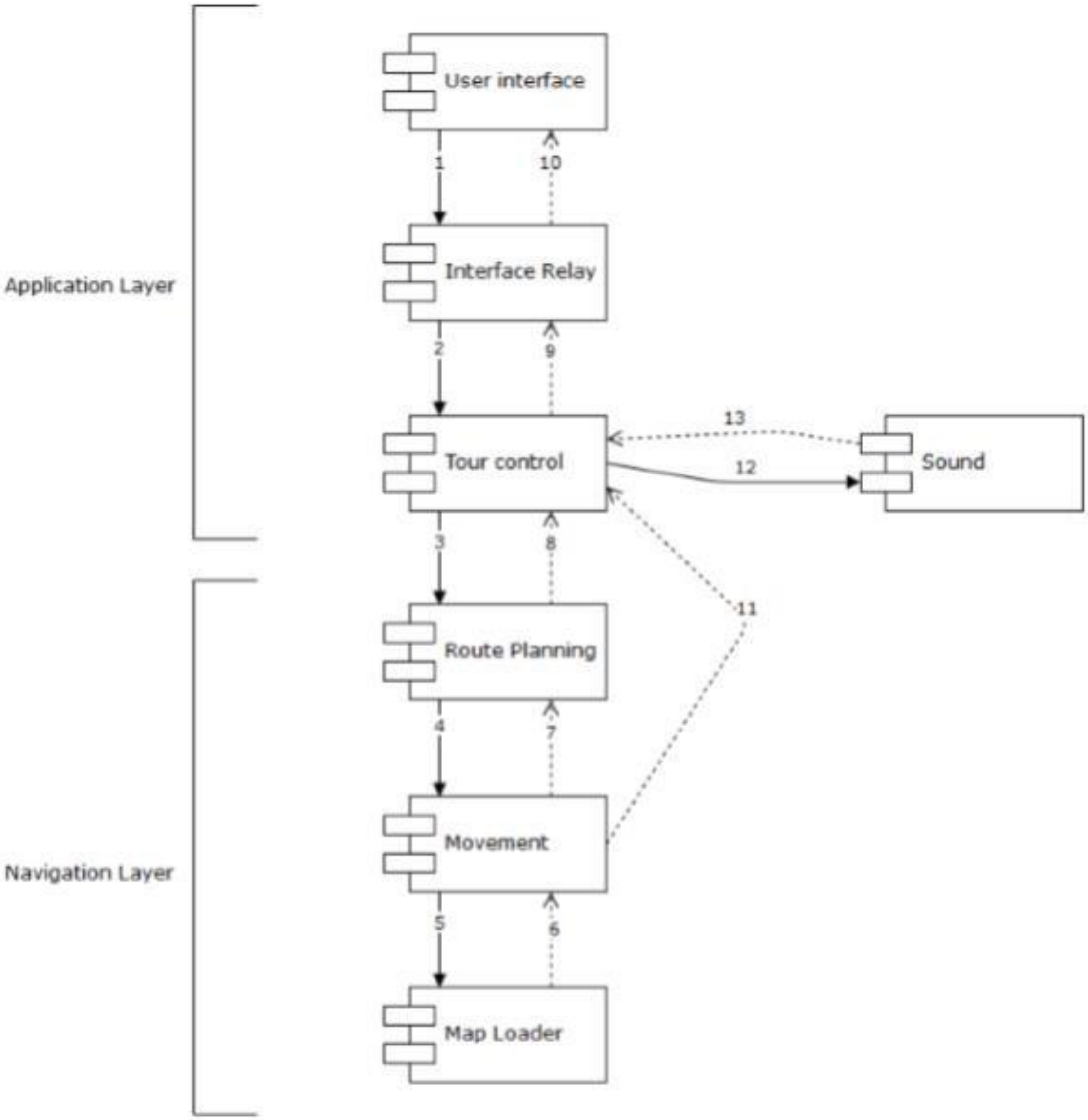


Fig 6. *Data flow through modules [11]*

3.2.1: The User Interface module is a module that deals with the transfer of information to and from the hand-held device included in the full TTG project. This module, will however, be discussed in a different report. This module is designed based on the Model

View Controller (MVC) architecture and is the TTG's way of interacting with the users. The interface passes information on the places that the user wants to visit to the mobile-relay module. The user-interface module also receives information from the modules on the workstation and Turtlebot and displays it to the user when necessary; this includes information about where the user is in a tour, some information about the points in the tour, and others.

3.2.2: The mobile-relay module receives information on the points that are going to be traversed during the tour from the user-interface module and then transmits the information to the required modules. The mobile-relay module only serves as a pathway for the user-interface and the Turtlebot and workstation to communicate. It connects these two main parts of the TTG project using a socket connection (sockets are what allow two different machines to communicate) and in this case, the mobile-relay module uses a socket to run a communication service.

In the reverse, the mobile-relay module also transmits notifications from the Turtlebot and the workstation to the user-interface.

3.2.3: The Tour-control module can be likened to a centralized system that manages the running of some other modules. It serves as the logic center of the TTG and sends the appropriate information to the appropriate module, based on what is needed to get the TTG to run. It decides on when to play a sound, which module to send what kind of data to, and decides on how to move the Turtlebot.

3.2.4: The Route-planning module is the module in charge of planning the routes for the TTG to follow. This is an important part of the project since this is the module that counters the inefficiency problem in human tour guides. From previous work, this module developed a route based on the shortest path between the points or places involved. It models the problem as a Travelling Salesman Problem (TSP) and uses the Dijkstra's algorithm to compute the shortest part between pairs of points.

In the current project, this module has been modified to include path-planning algorithms that develop routes based on priorities values and time-limits. This algorithm takes in the amount of time that a user has for the tour and then creates a tour that fits in as many areas with the highest priority as possible. This way, users with little time for a tour can still be able to have a tour of the best or most important places in the school. The priority values were attained during the requirements gathering process.

3.2.5: The Movement Module is the module that controls the movement of the TTG. This module helps the TTG to load the required maps, localize the Turtlebot and move accordingly.

3.2.6: The map-loader module oversees loading the required map for the tour and switching between maps during the tour.

3.2.7: The sound module is also in charge of playing some sounds to welcome and describe certain locations during a tour to improve the interactivity of the TTG. The sound module is not a chatbot or conversational system.

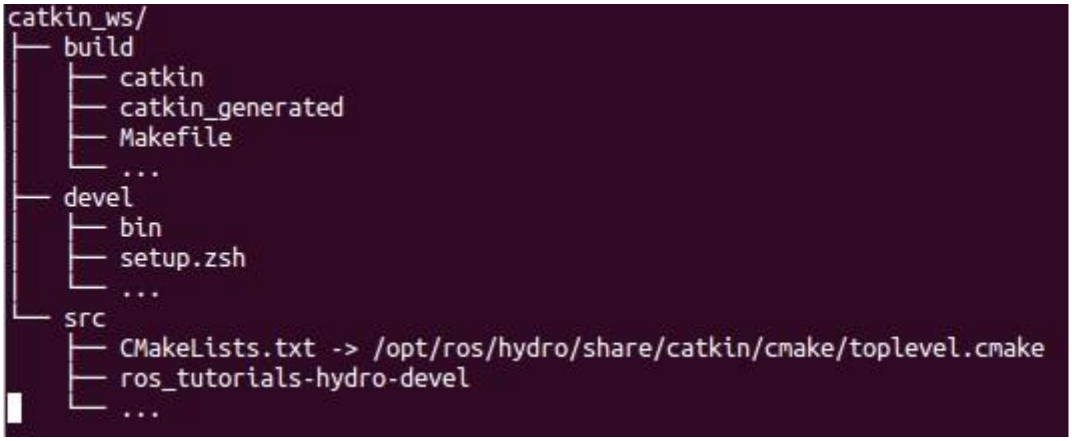
Another modification added to the previous work on the TTG is the **start-up module.** All the work needed to successfully start up the TTG is automated by the start-up module. The module contains 2 script shells, one for the Turtlebot and another for the workstation, that execute commands needed to source the required ROS environment, start the Turtlebot, run the scripts and do all other things (like configuring variables and the environment) that are needed to make the TTG run. This is important because the TTG should be able to run without the presence of someone with technical know-how. This allows just about any authorized person to easily start the TTG. From previous work, starting the TTG required running approximately eight terminals, each requiring sourcing.

3.3: Storage and Assets

The loader and sound modules are stored on the Turtlebot whilst the mobile-relay, pathplanning, tour-control and movement module are stored on the workstation. The maps for the Turtlebot and the sounds to be played by the TTG are stored on the Turtlebot. A JSON file, containing information on the IDs and co-ordinates of all the points in the map is stored on the workstation.

The TTG uses a **catkin-workspace** and project to organize and manage its files. Catkin is a ROS functionality that makes it easy to manage packages. Packages are containers for scripts, texts, libraries, datasets, assets etc. that are needed to run any module. “The goal of these packages it to provide this functionality in an easy-to-consume manner so that software can be easily reused” [7]

Catkin is a package that makes it easy for users to manage and build their own packages. It is a more efficient version of ROSBuild. Catkin Workspaces are a place to manage all of the different projects or packages that might be running simultaneously. The basic structure of a Catkin workspace is given below:



```
catkin_ws/
├── build
│   ├── catkin
│   ├── catkin_generated
│   ├── Makefile
│   └── ...
├── devel
│   ├── bin
│   ├── setup.zsh
│   └── ...
└── src
    ├── CMakeLists.txt -> /opt/ros/hydro/share/catkin/cmake/toplevel.cmake
    ├── ros_tutorials-hydro-devel
    └── ...
```

Fig 7. *Catkin workspace structure [7]*

The Source directory in the catkin-workspace is where the packages created are stored and managed. It contains all the scripts, datasets, service records, etc. needed to run the module.

The Build directory is where configuration, cached information and other temporary data are stored.

In the **Development directory**, test-runs can be executed on scripts before the system is fully deployed as functional.

The modules on both the workstation and the Turtlebot are managed by Catkin workspaces.

3.4: Technologies

3.4.1: ROS – Robot Operating System (ROS), is as the name implies, an operating system for robots. However, it is not entirely an operating system and serves more as a system that provides the functionality needed to work with robots. According to Generation Robots,

> "It provides not only standard operating system services (hardware abstraction, contention management, process management), but also high-level functionalities (asynchronous and synchronous calls, centralized database, a robot configuration system, etc.)" [8]

Both the laptop of the Turtlebot and the workstation have ROS installed.

3.4.2: Ubuntu

Ubuntu is a Linux Operating System and a Debian-based distribution. It is open-source and free and is arguably the best platform for developing and working with robots.

> “Ubuntu includes thousands of pieces of software, …, and covering every standard desktop application from word processing and spreadsheet applications to internet access applications, web server software, email software, programming languages and tools …” [9]

The Ubuntu distribution also provided a variety of packages that are essential to developing the modules needed for the TTG.

3.4.3: Groovy

Groovy is a distribution of ROS. “ROS Groovy Galapagos is the sixth ROS distribution release and was released December 31st, 2012.” [10] The TTG uses ROS Groovy.

Chapter 4: Implementation

4.1: Prior Work

Below is a recap of what has been done in previous work.

4.1.1: Path-Planning:

Previous work on the project used a Dijkstra’s algorithm to plan a path between points in the tour. The Dijkstra’s algorithm is an algorithm that uses heuristics and cost functions to determine the shortest path between points. This was the only algorithm available in the pathplanning module.

4.1.2: Mapping:

The maps available from previous work are limited to the office and classes spaces of the host university. The mapped area accounts for approximately 10% of the entire school space that needs to be mapped for effective tours by the TTG.

Also, the maps available for the different points in the region specified above are not the best. Because mapping was done with the Kinect sensor already available on the Turtlebot, borders that are not vertically projected are difficult to define. For example, the Turtlebot, during a mapping process, will map lawns as a traversable area – which means that during the tour, the Turtlebot might lead tourists through the lawns. The only things that count as borders for the Turtlebot are walls or pillars (with significant vertical magnitude). In essence, maps available for the area stated above do not have well defined borders or blocks.

4.1.3: Start-Up:

The system from previous work requires a lot of processes before start-up is initialized. These include, sourcing environments, bringing-up the robot and running all the scripts available for the different module.

4.2: Environment and Network Configuration

This section discusses the processes involved in starting the TTG project on a fresh laptop and Turtlebot with no prior configurations tuned to suit the TTG project.

4.2.1: Installation of Ubuntu and Groovy

To do this, a bootable pen drive with the latest version of Ubuntu (Bionic Beaver 18.04) was created and then run on both the Turtlebot and the workstation to install the operating system. Eventually Ubuntu was downgraded to 12.04 (Precise Pangolin).

Previous work was based on ROS-Groovy. However, because ROS has newer distributions than Groovy, Groovy is not supported on more recent versions of Ubuntu. Again, support for the Turtlebot and documentation on using the Turtlebot are readily available for Groovy than other ROS distributions. Hence, it was necessary to install Ubuntu 12.04 rather than latest version of ROS so that there could be enough support for both Groovy and the Turtlebot.

4.2.2: Network Connectivity

This process is about getting both the Turtlebot and the workstation to communicate effectively. To do this, the two systems needed to be configured so that they could work in tandem. Before information can be passed through the two systems, they needed to be registered on a local network. This local network was usually provided by a hotspot.

To allow the modules to work together and call each other regardless of the machine they resided on, the systems needed to be connected in ROS as well. To create this connection, a ROS master was set. The ROS master is the main system in the connection and "provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another." [8] The Turtlebot was set to be the ROS-master. The variables for the IP for the ROS maser and workstation were inserted in the source bash file (bashsrc) and this enabled the connection between the Turtlebot and the workstation, as it related to publishing and subscribing to nodes and services on either system. The network was tested in a tutorial available on husarion.com.

4.3: Catkin Workspaces and Project

These were how the different modules and projects were organized and managed so that they could be run with ease and simultaneously. The code available from previous work had to be migrated into a catkin workspace. However, because previous work had already structured a catkin workspace for the project, the work had to be migrated, built and compiled for it to run.

In essence, there was no need for the catkin project to be recreated. Some of the resources for some functionalities called on by the modules in the TTG were not available in previous sources. For example, the stds-msgs ROS package that allows messages that are transmitted by the various ROS nodes to be described (this helps the different modules understand and work with the data they receive better), was not accessible on ROS Pack – which is a package in ROS that manages all the different packages installed on a ROS machine. As a result, a git repository with the needed packages needed to be cloned into the catkin workspace to enable the stds-msgs package and others to work as required.

4.4: Miscellaneous

Script-Specificity

Some of the scripts needed to run the different modules included information that were machine-specific. This means that if the scripts were not run on a certain machine, the program crushed. The scripts were modified and all specific information relating to a particular machine were generalized. Information like machine names, port numbers and IP addresses had to be modified to ensure that the scripts could run on all machines – improving the reusability of the system developed.

Packages

Additional packages and services needed to be installed to support the different processes being run in the modules. Some of these packages were not available on the default package manager in Ubuntu(apt) and so needed to be installed via other package managers like

‘aptitude’. Some of the services that needed to be installed include: the Turtlebot service, Turtlebot-bring-up package, openssh (an open secure shell connection that allows different machines to run commands on other machines securely), etc.

4.5: Path-Planning Algorithm

The path planning algorithm that was added to the path-planning module from previous work is one that defines a path through set points based on the priorities assigned to the different points and the time available. Like stated earlier, this path planning algorithm uses the shortestpath algorithm to develop routes between points, making sure that the places with the highest priorities are considered first whilst staying within the time limit. In the end, this path planning algorithm produces a path for the tour that will last for the time the user has, whilst ensuring that the user sees the best of the school based on the priorities he inputs or the default priorities available. The core of this algorithm is dependent on the time and priority parameters and the process taken to get both of them is explained later in this chapter.

The diagram below shows the basic logic structure of the algorithm:

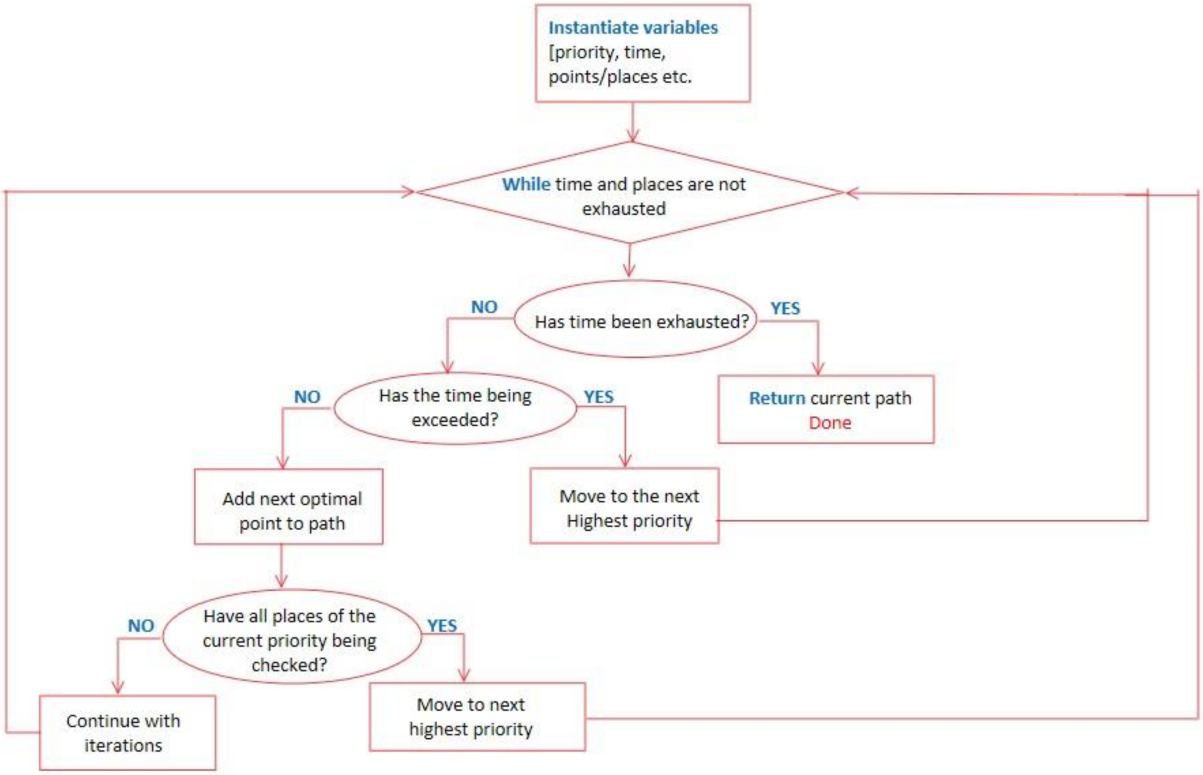

Instantiate variables
[priority, time,
points/places etc.
While time and places are not exhausted
NO
Has time been exhausted?
YES
Return current path
Done
NO
Has the time being exceeded?
YES
Add next optimal point to path
Move to the next Highest priority
NO
Have all places of the current priority being checked?
YES
Continue with iterations
Move to next highest priority

Fig 8. *Logic structure of priority-based path-planning algorithm*

The Pseudocode for the algorithm is given below:

```
def priority_planner(points_list, time_available,
start_point):        prioritized_list =
sort(points_list).delete(start_point);             start =
start_point;          cur_priority = 5;
```

```
        sub_list = list(for x in prioritized_list if x.priority == cur_priority)
          cur_time = 0;
cur_points = start;


          while cur_time <time_available && cur_priority != 0:
paths = list(path(cur_points & point) for point in sub_list);
times = list(time(path) for path in paths);


                 if (min(times) == time_available):
                           cur_time = min(times);
          cur_path = min(times).path;


                    elif min(times) > time_available:
          cur_priority = cur_priority - 1;
                          sub_list = list(for x in prioritized_list if x.priority == cur_priority)
                          continue #jump to next iteration with new sublist


                  else:
                              cur_path = min(times).path
                    cur_time = min(times);
  sub_list = sub_list.delete(newest_point) #where newest point is the latest point added to the mini
tour


                              if size(sub_list) == 0: #checks if all the points of a certain priority has been
checked then moves to the next priority
```

4.5.1: Explanation of the Algorithm:

The algorithm takes in the list of the points to be traversed (which is gotten from the user-interface and then the mobile-relay module), the time the user has for a tour and the start point of the tour (usually the current location of the user) as parameters. It then initializes variables like the current priority (sets it to 5 – which is the highest priority any place can be assigned), current time that it takes to complete a tour, and the current points needed to traverse. It then enters a loop that continually checks to make sure that the time available for the tour has not been exceeded or that there are still priorities to be traversed (the priority level should not be less than 1, since 1 is the lowest priority a place can have). Inside the loop, it continually maps the shortest path between points with the highest priority and shifts to a lower level of priority when needed. It exits the loop when the time available for the tour has been totally exhausted or when there are not more areas to tour.

4.5.2: Function to predict time

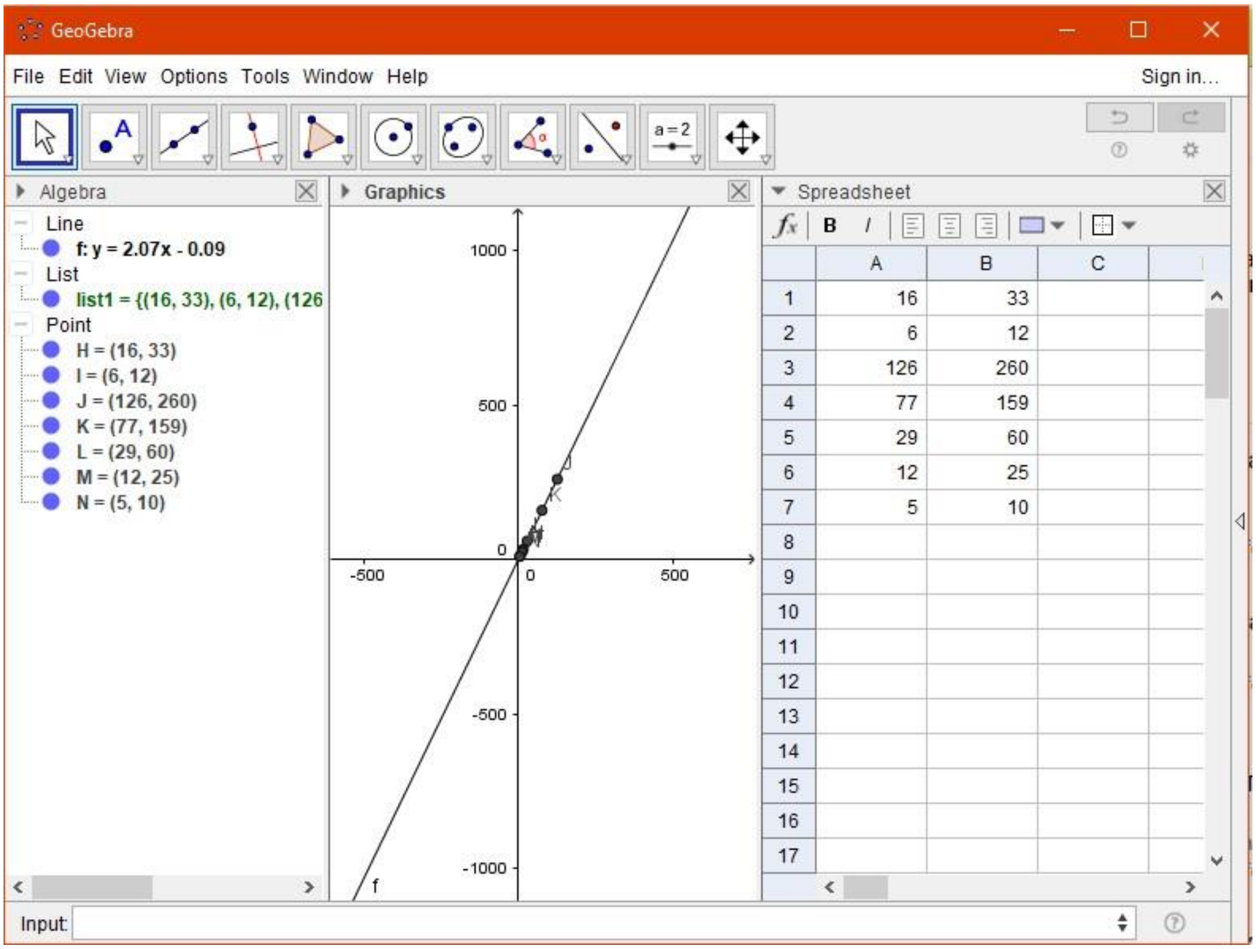
For the algorithm to work, it is necessary that the system is able to predict the amount of time it will take the Turtlebot to move between two points. However, there was no functionality in ROS or the Turtlebot that was readily available to solve the problem.

One way to find estimates for the amount of time it takes the Turtlebot to move from a point to another was to manually measure it – this meant that the Turtlebot would have to be taken through all points of interest in Ashesi University, then for each of these points, the distance to all other neighboring points will be calculated by driving the Turtlebot to that point and recording the time it took. This is not only tiresome but also inefficient; it is unrealistic to measure the distance between all possible points in any given map, especially when the map is large and spans over more than 50 acres of land. This method will also take a lot of time to complete and as a result defeat the purpose of automation – which, simply, is to make things easier.

Instead of the method mentioned above, another was used to estimate the time it took to move between any two points. Although functionality to estimate the actual distance (real-life distance) of the points in a map was not available, all the maps generated by the turtlebot have distance values that are relative to the size of the virtual grid that the turtlebot used in mapping the concerned area. This grid is usually shown to overlay the actual area and helps the turtlebot get the co-ordinates of any point in the map. Although these distances had no direct bearing on the real-life distance between the points, it was consistent enough to be used in modeling a predictive function (these are functions that use past or available information to estimate future events).

To get the predictive function that could use the distance information provided from the maps to estimate the time it will take to traverse the points concerned, the actual time that it took to traverse the points were also recorded. Hence, for 10 pairs of points on the map, the actual time it took for the Turtlebot to move between each pair of points was recorded. The Turtlebot was set to move at its lowest and default speed which is 0.2m/s. The lowest speed of the turtlebot was used because sometimes the turtlebot slows down either because of battery power or because of a weak network signal. In either case, using the lowest speed accommodates for most of the things that could go wrong and caters partly for the worst-case scenario.

After getting this information, the data was passed through the GeoGebra application which is a statistics and algebra software that allows users to model functions as well. The different points in the map were inputted with the distance from the map as x-coordinates and the recorded time it took to move between the points as y-coordinates. After plotting the points, the fit function was used to develop a linear function that represented the data well.. **Figure 9** shows how the function was developed from the data in GeoGebra. Note that, the image shows 7 points instead of 10 because some of the points were the same.



Line
f: y = 2.07x - 0.09
List
list1 = {(16, 33), (6, 12), (126
Point
H = (16, 33)
I = (6, 12)
J = (126, 260)
K = (77, 159)
L = (29, 60)
M = (12, 25)
N = (5, 10)

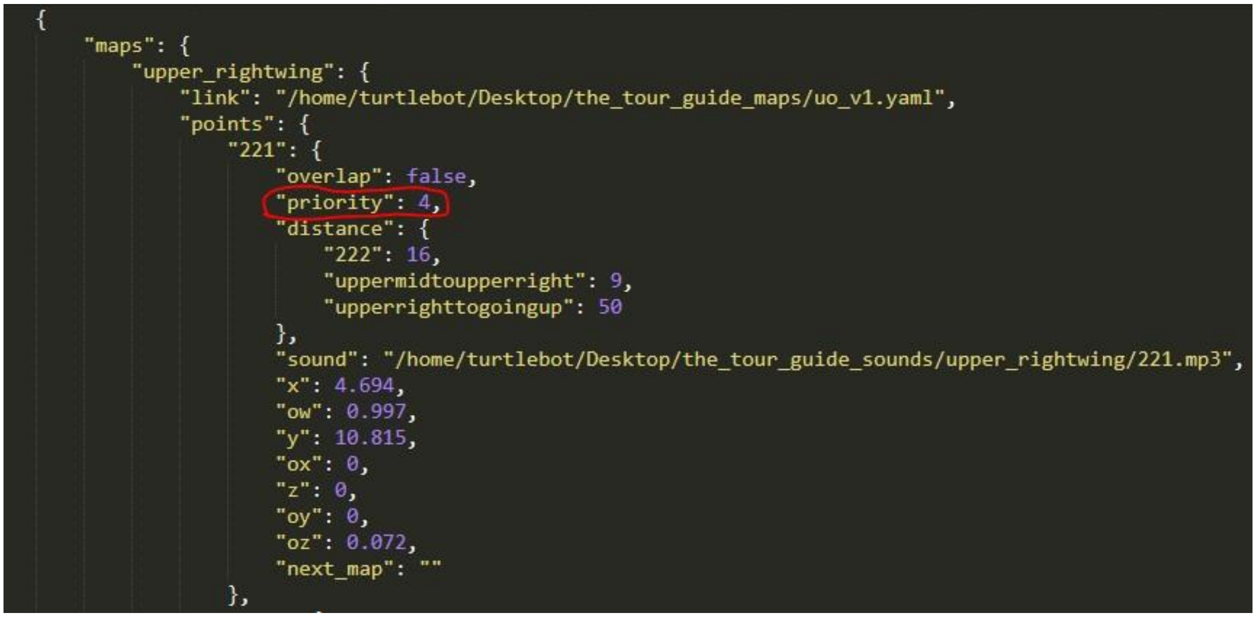
	A	B	C
1	16	33	
2	6	12	
3	126	260	
4	77	159	
5	29	60	
6	12	25	
7	5	10	
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			

Fig 9. *Modeling the time function*

The results from the modeling process produced the function: $y = 2.07x - 0.09$ where x is the distance returned from the map and y is the seconds it takes to traverse the points in seconds. Using this function, it was easy to calculate the time it took to traverse the point which was needed for the priority-based path planning algorithm.

4.5.3: Priorities

Aside from predicting the time it will take to move between any pair of points, getting the priorities for the system is also essential. The priorities – which show how important a point in the map is during a tour – are stored in the JSON file. From previous work, metadata for each of the maps and the points within the maps were stored in a JSON file. The metadata included information on the ID of the point in the map, the position and orientation co-ordinates of the point, sound files for each point and similar others. For the purpose of the priority-based path planning algorithm, another metadata detail was added to the JSON file for each of the points in the map. This addition was labeled ‘priority’ and was assigned a value between 1 and 5, where 1 meant that the point was not an important place to see during a tour and 5 meant that the point was a must-see area during a tour. The different priorities for each of the points were gathered by speaking to students and the tour-guides of Ashesi University, as was detailed in the Requirement gathering section. **Fig 10** below shows how the priority metadata was stored in the JSON file.



```
{
    "maps": {
        "upper_rightwing": {
            "link": "/home/turtlebot/Desktop/the_tour_guide_maps/uo_v1.yaml",
            "points": {
                "221": {
                    "overlap": false,
                    "priority": 4,
                    "distance": {
                        "222": 16,
                        "uppermidtoupperright": 9,
                        "upperrighttogoingup": 50
                    },
                    "sound": "/home/turtlebot/Desktop/the_tour_guide_sounds/upper_rightwing/221.mp3",
                    "x": 4.694,
                    "ow": 0.997,
                    "y": 10.815,
                    "ox": 0,
                    "z": 0,
                    "oy": 0,
                    "oz": 0.072,
                    "next_map": ""
                },
```

Fig 10. *Assigning priorities*

After providing such information in the JSON files, the priority values could be derived for each of the points in the map using simple call statements.

4.6: Module for Automating TTG Start-up

This section has to do with implementing the module that does all that is necessary to start the Turtlebot Tour Guide. Because the TTG will be used outside the classroom setting and by a team (the team in charge of tours in Ashesi University) that is not expected to have the technical know-how needed to run scripts and source environments, as is required for the TTG to start, it is important that the process of starting the TTG is simplified. To simplify the process, two shell scripts were created – one for the turtlebot and the other for the workstation. Shell scripts are files that contain a list of commands that are executed when the file is run.

The shell script on the turtlebot contains a command to source the environment – sourcing helps you run the commands that are needed before your program can run in a terminal or shell -, load and run the 'loader module' which deals with finding the right map for a tour and the 'sound module' which plays a sound during the tour and then runs the command needed to bring-up or awaken the turtlebot. It was noticed that one of the ports needed for the bringingup of the turtlebot always had another process running on it, by default, after start-up. This became a hindrance to using the turtlebot since the bring-up was rarely successful. To solve this problem, a command that checks and frees the port (i.e. kill the processes running on the port) was added to the shell script. It checks if there are any processes running on the port needed for the turtlebot awakening. If there are any processes, the ID of the process(es) running on the port is returned and another command is issued to kill the process or stop the process from running so that the turtlebot bring-up process can use the port instead. This ensures that after this single shell script is run, all the processes and packages that need to be running are started successfully and are ready to be used by the TTG.

The shells script on the workstation does something similar to what is done on the turtlebot: It firstly sources the terminal, then it pings the ROS_MASTER (the turtlebot itself) to ensure that there is an active network connection between the workstation and the turtlebot. After this is confirmed, a command that launches all the modules on the workstation – like mobile-relay module, the movement module, etc. is executed. This also summarizes the initialization of the TTG from the workstation.

With these shell scripts, people who operate the TTG can easily start a tour with or without the presence of someone with the technical know-how needed to run the TTG.

4.7: Mapping

In this section, we discuss the implementation of the mapping functionality of the turtlebot. This process required two main activities – improving upon the quality of the already existing maps developed from previous work and mapping new areas that can be traversed during a tour.

4.7.1: Improving Map Quality

From previous work, maps of some areas in Ashesi University were created. These maps were collated and used for the TTG. However, most of the maps that existed were of poor quality. The maps had unclear borders or obstacles which made it difficult for the turtlebot to move within the map as expected. The sensor that is built into the turtlebot is a laser sensor and so, it does not map items with no vertical projections as borders. For example, when mapping an area using the turtlebot, walls and pillars can be mapped well as borders. However, downward stairs and lawns will be mapped as areas that the robot can pass through. This is not ideal since the robot is not physically equipped to move down stairs and because it will be inappropriate to take visitors of Ashesi University through a lawn during a tour. Due to the limitations of the sensor in the turtlebot, these issues were previously left unsolved although recommendations from previous work suggest that the quality of these maps be improved.

Fig 11 is an image of a map generated by the turtlebot during a typical mapping process.

Mapping with the turtlebot simply involves running the ROS Gmapping package - which allows the turtlebot to localize itself in the map (i.e. know its position in the map) whilst simultaneously mapping the area in a process known as SLAM (Simultaneous Localization and Mapping) – and the RViz package which is a simple interface that visualizes the mapping process by taking the information given by the sensor of the turtlebot and building models to replicate it as it is in real life. After running these packages, mapping is done by driving the turtlebot around the area thoroughly using teleoperation – a process that allows a user to control the turtlebot as with a remote control. Afterwards, the map can be saved as a '.pgm' (portable gray map) file. This file shows a diagram of the mapped region in black and white, where black areas represent borders and white areas represent obstacle-free areas or areas where the robot can move freely.

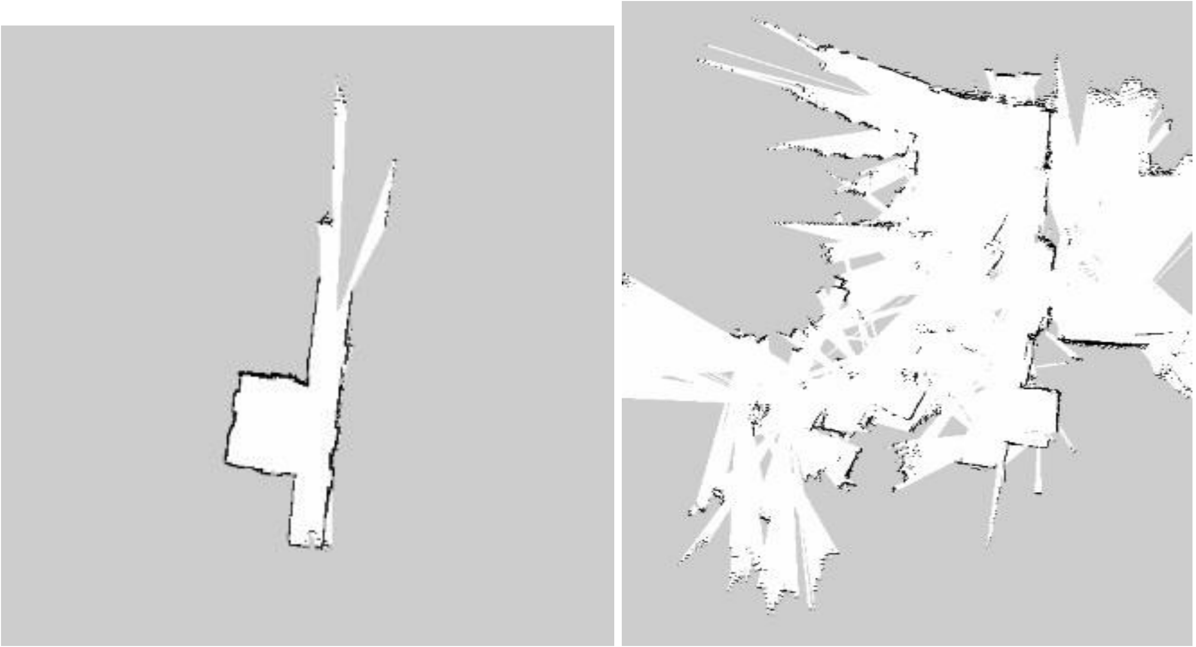


Fig 11. *An example of a turtlebot-generated map* **Fig 12.** *Turtlebot-generated map*

Figure 11 shows a map that has fairly well-defined borders that will make it easy for the turtlebot to move through the map. However, most of the maps generated are not like this and are ambiguous with what they show as obstacles and otherwise, as is in figure 12. This image is of a corridor with a few corners and a lawn around it. Figure 12 is a good example of a map with low quality that could cause problems during a tour.

However, a solution was found for this problem in this project. Using Gimp, one of the few image editor software that can read files with the '.pgm' extension, the borders and traversable areas could be redefined to produce a clearer image for the turtlebot as is shown in Figure 13 below.

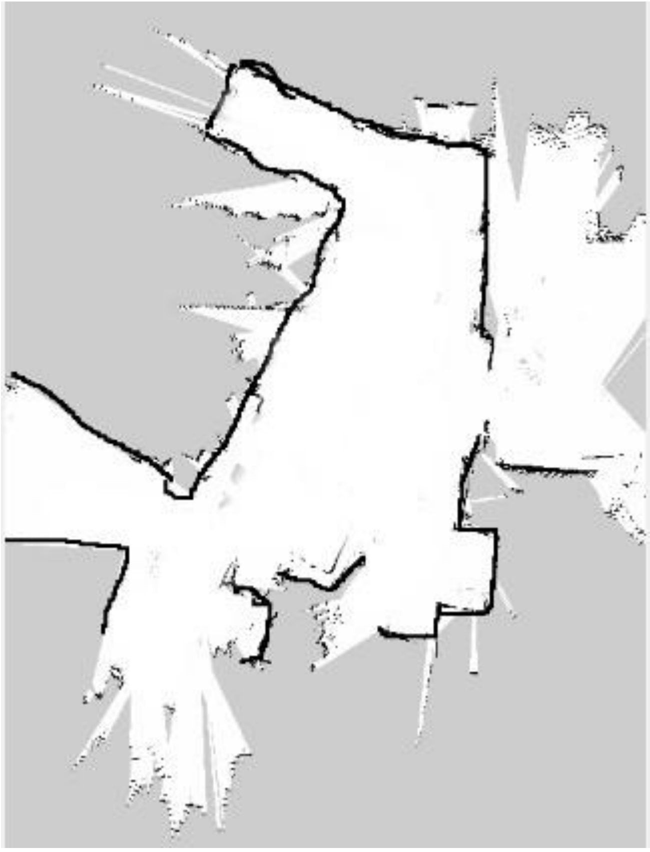


Fig 13. *Edited turtlebot map*

Although figure 12 and 13 are maps of the same area, the stark difference between them is obvious and this difference is reflected when the turtlebot navigates the map as well. In the old unedited map, Fig 12, the turtlebot sometimes makes turns in some parts as though there were an obstacle even though there were none in real life. The turtlebot also tried moving through the lawn several times to get to its destination. However, with the GIMP edited map (fig 13), these issues are rarely seen since the turtlebot has a better sense of what is an obstacle and what is not; where it can move to and where it cannot move to. The map was edited by simply drawing black lines over the areas that are actual borders and erasing the gray color from the areas that were traversable. Editing the turtlebot-generated maps with GIMP is a simple way to solve the problem of poor map quality.

4.7.2: Creating new maps

The TTG requires new maps because more infrastructure and space has been added to the campus of Ashesi University. For the tourists or visitors to have a good tour experience, it is necessary for them to see all the new parts of the school as well. Generating these new maps involves the simple process of mapping using Gmapping and RViz, as explained early on. After these maps were generated, those that were not well mapped were edited with GIMP to make them better.

For each new map generated, metadata on the map and the points within the map are entered into the JSON file. The structure for the JSON file was developed from previous work and contains parameters like:

‘link’ – this provides information on the path to the map’s file.

‘points’ – this provides information on all the points that are within the map.

‘overlap’ – this stores a Boolean (true or false) that tells if the current map can be directly linked to another

‘sound’ – stores the link to the sound file for the points in the map

Orientation and position coordinates

For each map that was generated or created using the turtlebot, the map had to be stored in the right directory and the metadata for the map had to be entered into the JSON file so that the TTG could navigate those new areas as well. The orientation and position co-ordinates help the turtlebot know where it is in the map and in which direction it is facing. During mapping, this information was derived by echoing the data being transmitted from the AMCL ROS topic – it publishes information on the orientation and position of the turtlebot at any point in time and is known in full as the Adaptive Monte Carlo Localization topic.

After getting the maps for the new areas, editing them where necessary and filling the metadata information on each of them, the TTG was able to move through them successfully.

4.8: Alternative Mapping Method

Although the maps generated by the turtlebot have problems, they still are good for simple navigation tasks. However, the process of mapping large areas takes a lot of time and are not very good. The existing maps from previous work are structured in a way that is similar to stitching smaller maps together to form a bigger one. This is made possible by the ‘overlap’ attribute of each map that shows which map it is linked to, if any at all. However, this concept causes the bigger map, made from smaller maps, to look linear and too simple. For example, a smaller map can be connected to more than 2 maps in reality but during this ‘stich-like’ mapping process, the smaller map will appear as though it was connected to only one map. This causes the bigger map to look like the smaller maps within it were only connected end to end. During navigation, this problem is evident when the TTG uses a path that is not the most optimal because it does not know the other paths that exist.

It is difficult to solve this problem using the turtlebot's mapping functionality. A good solution would have been to map the entire region of interest as one map, not a collation of smaller maps. However, in large areas, this is infeasible because the turtlebot sensors and mapping functionality are not able to work at such a large scale.

The one way to solve this problem is to create a custom map. This was done by getting a rough version of the floor plans of Ashesi University. After this, the actual distance between 2 linear points in the map, (both at extreme ends) was measured and used to scale the map so that the turtlebot could move through the map whilst knowing how far apart the points in the map were. This process has been used by other users of the turtlebot who have faced similar mapping problems as has been discussed here. According to an undergraduate research team, who created the program that was used to convert the floor plans of Ashesi University into a scaled .pgm file in this project, "If the user has access to a floor plan of the building, it could be converted into the appropriate "pgm" and "yaml" that the robot reads as a map; then the time consuming and lack of accuracy issues would be eliminated."[5]

Some of the other implicit functionalities implemented include security. To cater for the security of the network that was used for the TTG's communication, a hotspot network was used, with the WPA2 PSK encryption to ensure that the network was protected from people who might to try to hack into the system. The WPA2 PSK encryption type uses the Wi-Fi Protected Access 2 technology in addition to a Pre-Shared Key(PSK) to secure the network connection. For physical security, the Turtlebot will need to be stored in a room that could be locked and supervised appropriately.

In addition to the new modules and scripts created for the TTG, a problem resolution log was created. This log is a compilation of all the problems that were encountered whilst working on the TTG and how they were solved. The log contains errors that mostly come from deploying the previous work on a new system. Because the previous work was implemented on a specific machine that had some specific packages and programs running already, there were a lot of problems that were encountered when running it on a new Turtlebot and workstation. The problem-resolution log is provided in Appendix B.1.

Chapter 5: Testing and Results

In this section, the various functionalities that were implemented for this project were tested to make sure that they worked as expected. During the testing, a successful functionality was one that was reliably accurate – which, for this project, required that the functionality be accurate at least 8 out of 10 times (80% accuracy mark) -, performed all the tasks it was created for and worked well with the entire TTG system. The results from the test, because this project was developed using an iterative method, were always considered so that the programs were changed to suit the results of the test. The different tests run for the different functionalities and the results for each of them are given below:

5.1: Path-Planning Algorithm

Approach

Majority of the tests conducted to ensure that the path planning algorithm was implemented successfully was in the form of unit testing. Unit testing involves testing a single component or part of a system at a time. This was used for the path-planning algorithm because the results of the test could be readily seen after running just the algorithm – running the entire system was not necessary to see the effect of the algorithm. The route-planning module contained the path-planning algorithm and during testing, it was given an input of points to traverse. After receiving the points, the algorithm was called on the input and a list of points, in the order that they needed to be traversed was returned as the results of the test.

The priority-based path-planning algorithm was tested by first of all assigning the priorities that were derived for the points during the survey. This was imply done by adding the priority label to the JSON file and equating it to the priority value of the point in the map. After this, a random list of points in the maps were collated and passed to the algorithm as input, together with a start point and an empty list. This empty list was created to give the visitors of the university a chance to enter their own priorities for each of the points so that this, instead of the values in the JSON file, was used by the algorithm to plan the path. Because this was not a core of the system and was just a function that went to improve the experience, it was ignored during this test, which is why an empty list was sent as a parameter. This way, the algorithm used only the priorities provided by the JSON file to create the path.

Tests and Results

After the algorithm had run on the points that were given as input, it returned another list of points that were ordered to form a route. After these points were returned, the time it took to traverse all of those points was also returned. This was done so that the accuracy of the algorithm could be tested. When this was done, 100% of the tests provided a time value that was either less or equal to the amount of time that the visitor had for the tour – which went to show that, in the sense of regulating the time for the tour, the algorithm was reliably accurate.

Although the time factor of the tour had proved successful after the test, it was realized that there was not enough variation between the points that the algorithm returned to be traversed. It always seemed to be the same points and small changes made in the input were not reflected significantly in the output of the algorithm. After further consideration, it was realized that this problem was so because most of the points in had a priority value of 4. These made the priority values very monotonous and caused the algorithm to return paths that made it look as though all the points in the map had the same priority. Because of this problem, because too many points had the same priority value, the paths that were returned did not look that they were sorted based on priority.

Action Taken

To counter this problem, the range and step for assigning the priority values was changed. Initially, the priority values were ranged from 1 to 5. During execution, the algorithm was set to move a full step (1) lower to get the next priority value. This means that, after the algorithm had run through all the points with a priority of 5, it moved to all the points with a priority of 4. This step value was changed so that instead of moving to 1 less the current priority value, it moved to 0.1 less the current priority value and used a range instead. With this done, after running through all points with a priority of 5, the next lower level priority was going to be between 4.9. This change also had to be reflected in the JSON file that returned the priority values for each of the points. Rather than round the average from the results to a whole number, as was done earlier, it was rounded to one decimal place. This way, the priorities of the points in the JSON file were more different, making the priorities of the points more diverse and the strength of the algorithm more obvious. The table below shows the new list of priorities assigned from this test.

Table 2: *new priority values*

Area	**Rating(average)**
Library	3.9
Faculty Offices	3.0
Computer labs	3.7
Engineering labs	4.3
Norton Motoulsky hall	3.8
Cafeteria (Akonnor)	3.9
Ashesi Football Pitch	4.2
Health Center	4.4
Research Building	4.4
Student Hostels (Halls 1-8)	4
Student Hostels (Halls A-F)	4.2
Student Hangout Spaces	4.2
Fabrication Lab	4.2
Cafeteria (Big Ben)	3.8
Lecture Halls	4.1
Ashesi Shop	3.4
Ghana Climate Innovation Center	3.8
Off-Campus Hostels	3.2
Sports Courts	4
The Hive	3.9

5.2: Mapping

Approach

To test the maps that were derived for the TTG, sample tours were run through the maps. The first maps used were the maps that were edited using GIMP and the next map used was the map created from the floor plans of Ashesi University. The actions and navigation of the Turtlebot was recorded through each of these maps and used to validate the effectiveness of the system.

Tests and Results

After running the system on maps that were edited using GIMP, it was noticed that the Turtlebot found it much easier to move through the map and to make accurate turns. Unlike the unedited maps that were used from previous work, the robot did not make any abrupt turns in the middle of an obstacle-free area. This made the tour experience better and smoother.

When moving through the map generated from the floor plan, the Turtlebot sometimes had issues getting the distance between any two points right. It sometimes made a turn even before it got to where it was supposed to make the turn. This caused it to sometimes bump into walls or some obstacles or caused it to miss turns by a few meters.

Action Taken

After analyzing this behavior, the problem was attributed to the distance values that were given to the program that converted the floor plans into a map. These distance values were estimated since the actual distance was not readily available. Hence, the map was scaled using the erroneous distance values which caused the Turtlebot to move within a map that was virtually different from the one on the ground. To resolve this, the estimated distance values between the extreme points in the map were tweaked constantly and the system was run again.

However, the errors were still present, although with less frequency. Because of the less than optimal navigation through the maps created for floor-plans, the Turtlebot-generated maps were maintained as the main maps for the TTG.

5.3: Start-up module

Approach

The start-up module was tested in a simple and straightforward way. This is because the results of the test were easily derived: if the TTG did not start successfully, then the module was not working as required. Hence, to test the module, the shell scripts were simply run.

Test and Results

The first time the shell script was run on the Turtlebot, the Turtlebot bring-up process was rarely successful. There was always a process in the bring-up that was trying to run on a port that, apparently, was always been used by another port. This resulted in a ‘socket error’.

When the shell script was run on the workstation, it worked well as expected. It started all the modules accurately and functioned as was required. There were rarely any problems. However, there were a few times where the modules could not start and returned errors about the system’s inability to connect to the ‘ROS_MASTER’, which is the Turtlebot. When this happened, it was because there was no network connection between the workstation and the Turtlebot.

Action Taken

To solve the problem with the ‘socket-error’, an additional command that killed any process running on the needed port was added to the shell script. After this, 100% of the startups on the Turtlebot were successful when the shell-script was used.

On the workstation, a command was added to ping the Turtlebot or the ‘ROS_MASTER’ before the script tried to start running any modules or scripts. When the ping was successful, the scripts or modules were run and when it was not successful, a message asking for a network connection was printed out unto the terminal.

After running all these tests, it was decided that the various functionalities developed for this project were reliably accurate and appropriate for the system.

Chapter 6: Conclusion and Recommendation

The Turtlebot Tour Guide (TTG) project, which is a project that seeks to model a Turtlebot into a tour-guide for Ashesi University, was started a few years ago and has since produced a system that was partly functional. For the system to be fully complete and functional, mapping had to be done, a priority-based path-planning algorithm had to be added and a module that simplified the whole process of starting up the TTG had to be created. This project is about adding all of these things and ensuring that they were merged well with the system to product a functional whole. This project has successful enhanced and improved the system that was present from previous work by:

Making maps that existed from previous work more accurate and of better quality so that the TTG could better navigate the areas in Ashesi University.

Creating and adding maps of the new areas of Ashesi University to the tours given by the TTG so that visitors could see all the places in the school.

Including a priority-based path-planning algorithm that allows visitors to see the best parts of Ashesi University within the little time that they have available.

Creating a simple and effortless way to start the TTG by collating all the commands necessary for a successful startup of the TTG into shell scripts that can also be run by people without any technical expertise.

Hence, the TTG project meets all the requirements and contributions that it set out to achieve. Aside from this, the TTG has also introduced a new touring experience to people of

Ashesi University and in general, visitors of Ashesi University – which include people from all over the world. In light of this, the TTG has made it easy to gauge people's reception and reactions towards automated systems, like this one, which some people might find threatening.

However, this fully functional framework can be used to create even more complex systems. For example, a simple delivery system can be built off this tour-guide system. It is recommended that this basic navigation framework be used to create more sophisticated systems that could serve important purposes in Ashesi University and wherever this system can be used.

Also, to improve the interactions between the TTG and the visitors (human), it will be great to create a simple conversational system for the TTG that will allow it to have simple and seemingly natural conversations with the people it takes on tours. This will actively improve the touring experience and make the TTG almost as good as human tour-guides.

Conclusively, although this project has made previous work on the TTG more functional and accurate, the essence of the project can be enhanced if other systems that perform simple tasks are built off it. It is better for the TTG to become more than a tour-guide to Ashesi University.

References

[1] Mackay D. 2005. Path Planning with D*Lite - Implementation and Adaptation of the D*Lite Algorithm. Technical Memorandum. Defense R&D Canada – Suffield

[2] Buggard W et al.1999. Experiences with an interactive museum tour-guide robot. Artificial Intelligence 114,1. (October 1999), 3-55.DOI: 10.1016/S0004-3702(99)00070-3

[3] Feng X. and Liu W. (1997). Robot autonomous navigation. (June 1997). Retrieved December 12, 2018 from http://resolver.caltech.edu/CaltechCDSTR:1997.009

[4] Li A. et al. 2016. Active localization with dynamic obstacles. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE Xplore, 1902-1909.DOI: 10.1109/IROS.2016.7759301

[5] Andrew S. et al.2017. A Student Project using Robotic Operating System (ROS) for Undergraduate Research.(June 2017).Retrieved April 23, 2019 from https://www.researchgate.net/publication/325078294_A_Student_Project_using_Robotic_Operating_System_ROS_for_Undergraduate_Research

[6] (2019). Turtlebot. Retrieved February 28, 2019 from https://www.turtlebot.com/

[7] Fernandez E.2015. The ROS Filesystem levels (August 2015). Retrieved January 1, 2019 from https://hub.packtpub.com/ros-filesystem-levels/

[8] Mazzari V.2016. ROS - Robot Operating System.(March 2016).Retrieved April 1, 2019 from https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/

[9] What is Ubuntu? Retrieved April 1, 2019 from https://help.ubuntu.com/lts/installationguide/s390x/ch01s01.html

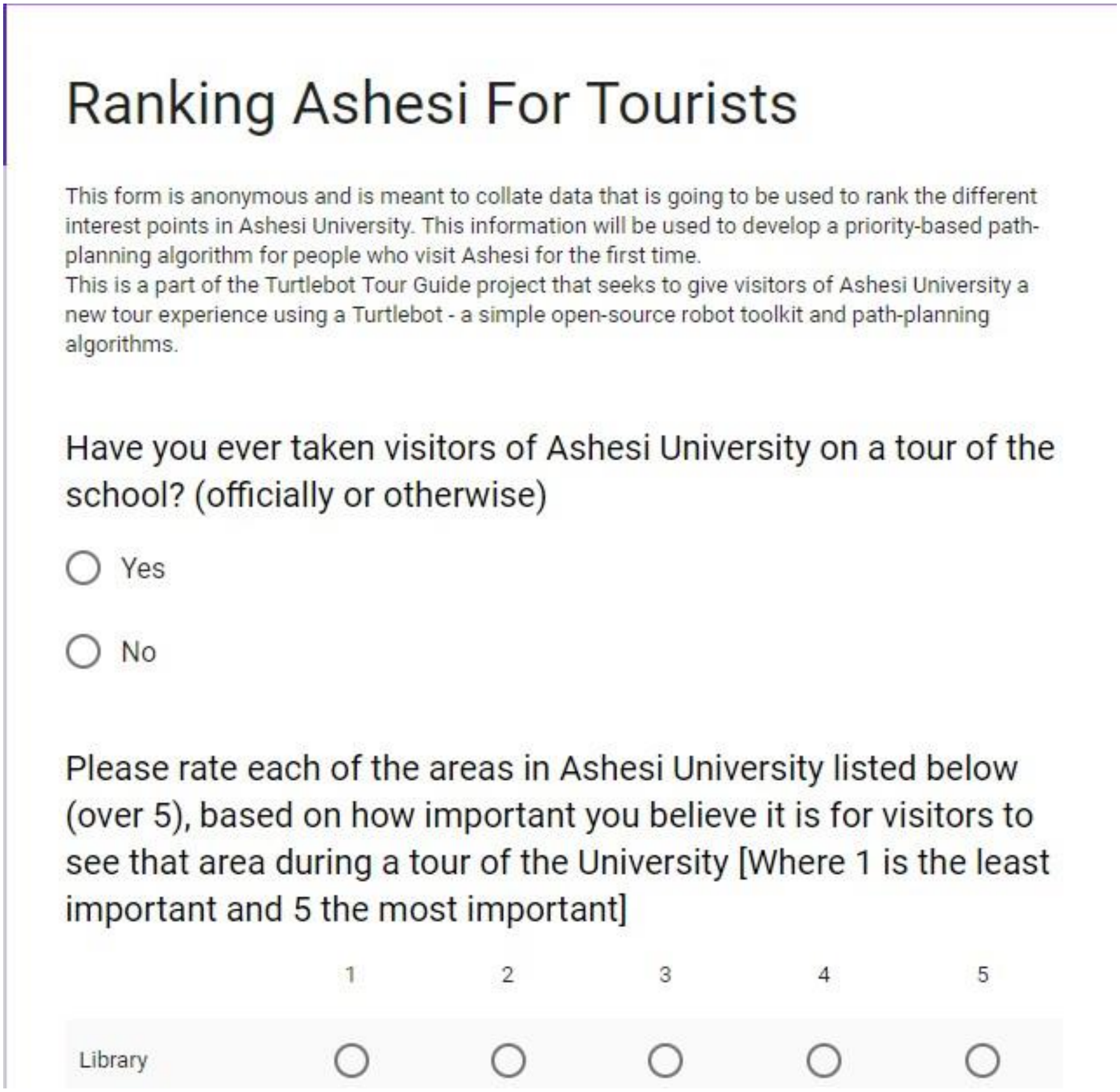
[10] groovy - ROS Wiki. Retrieved April 1, 2019 from http://wiki.ros.org/groovy

[11] Yebisi S. 2016.Tour guide robot. (April 2016)Retrieved December 19, 2018 from: https://air.ashesi.edu.gh/handle/20.500.11988/282

Appendix

Appendix A [Requirements Gathering]

A.1 Questionnaire



Ranking Ashesi For Tourists

This form is anonymous and is meant to collate data that is going to be used to rank the different interest points in Ashesi University. This information will be used to develop a priority-based path-planning algorithm for people who visit Ashesi for the first time.
This is a part of the Turtlebot Tour Guide project that seeks to give visitors of Ashesi University a new tour experience using a Turtlebot - a simple open-source robot toolkit and path-planning algorithms.

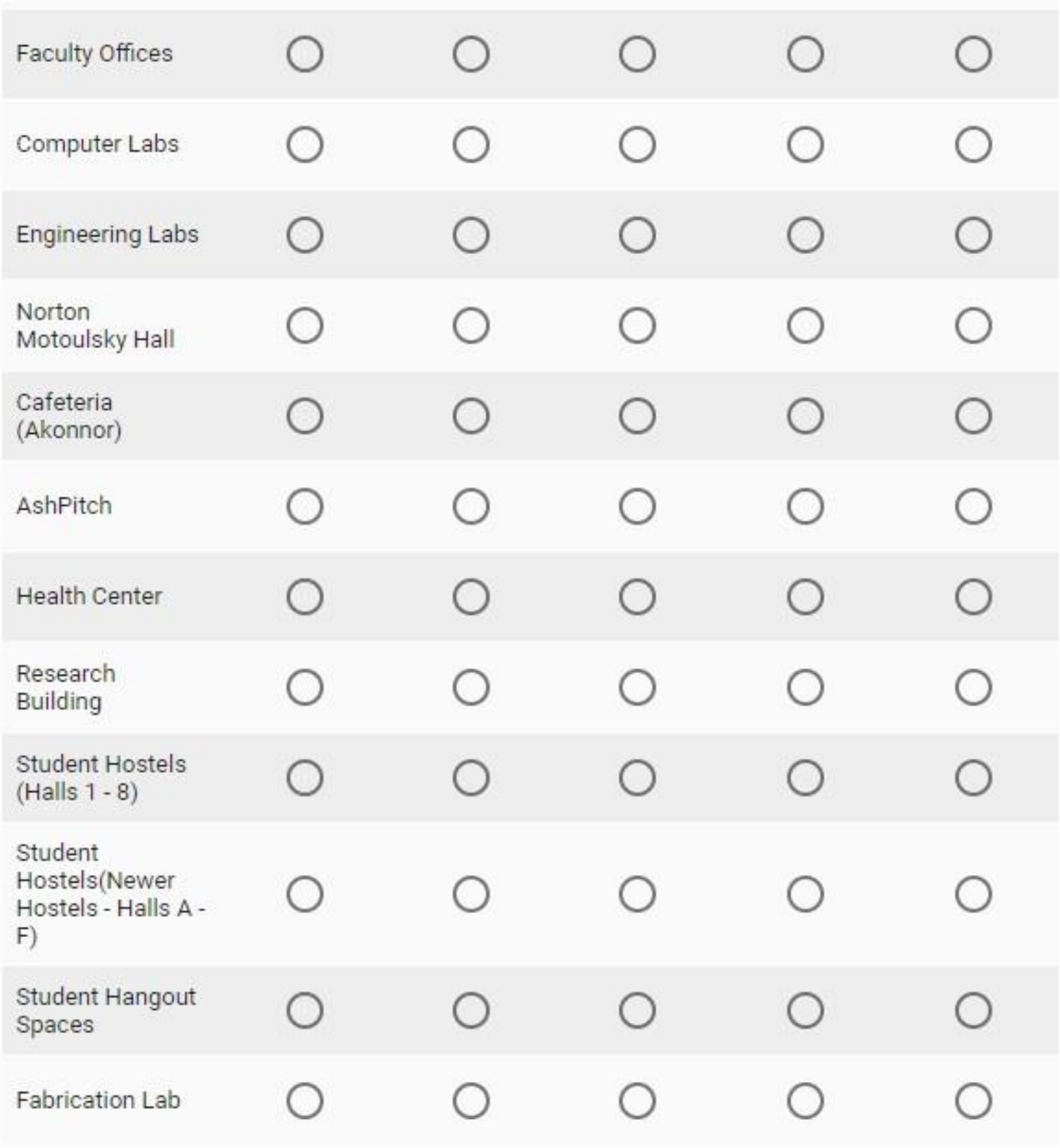
Have you ever taken visitors of Ashesi University on a tour of the school? (officially or otherwise)

○ Yes

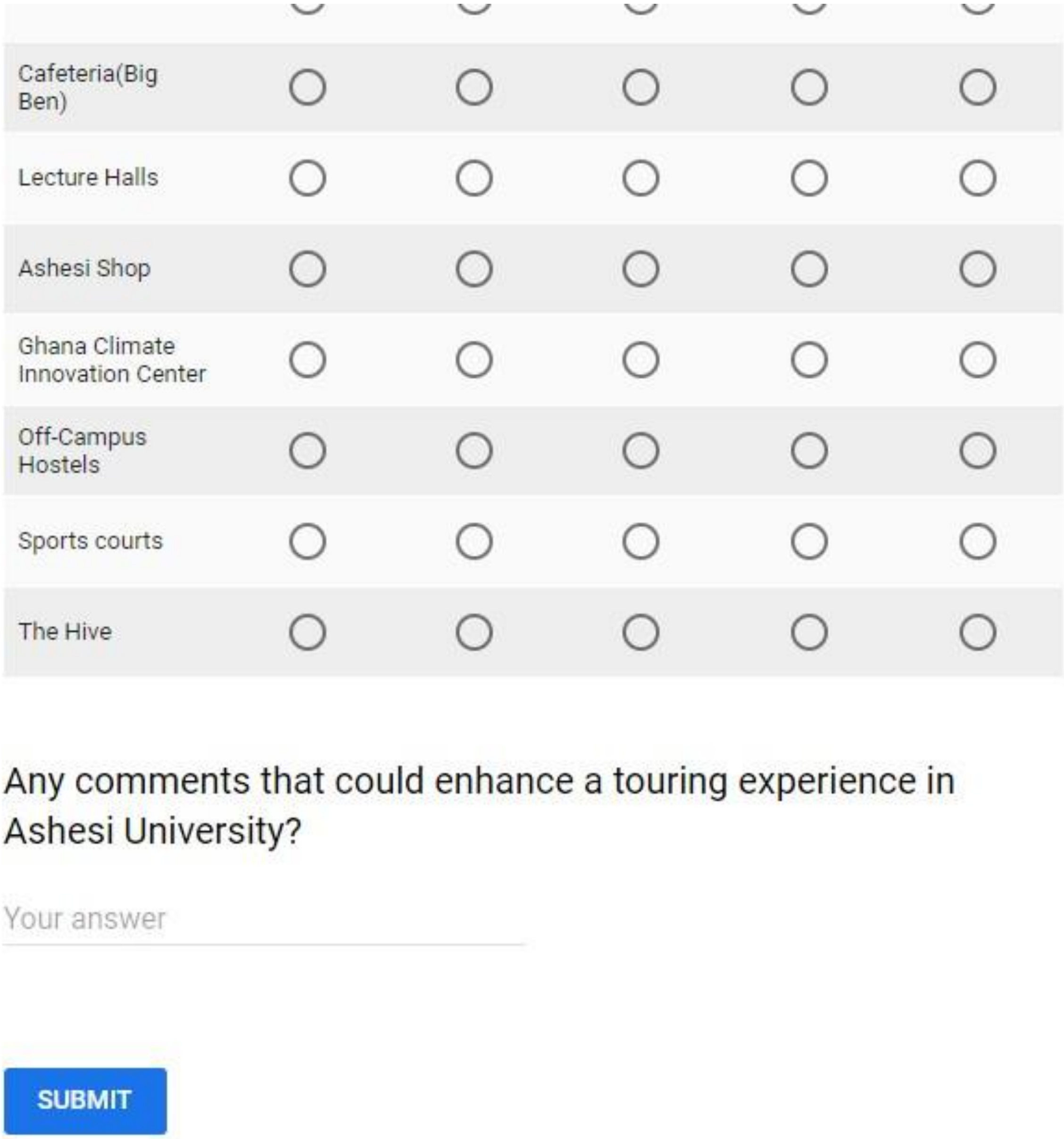
○ No

Please rate each of the areas in Ashesi University listed below (over 5), based on how important you believe it is for visitors to see that area during a tour of the University [Where 1 is the least important and 5 the most important]

	1	2	3	4	5
Library	○	○	○	○	○



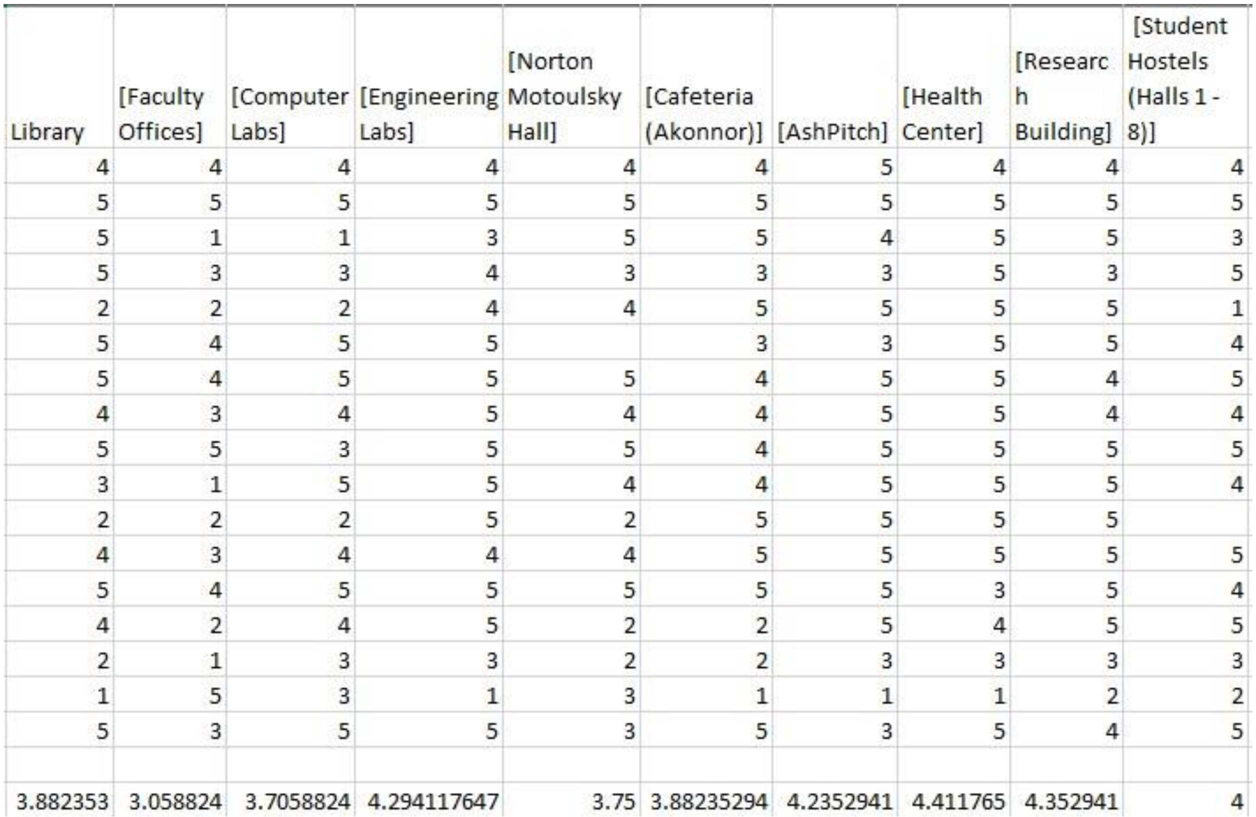
Faculty Offices	○	○	○	○	○
Computer Labs	○	○	○	○	○
Engineering Labs	○	○	○	○	○
Norton Motoulsky Hall	○	○	○	○	○
Cafeteria (Akonnor)	○	○	○	○	○
AshPitch	○	○	○	○	○
Health Center	○	○	○	○	○
Research Building	○	○	○	○	○
Student Hostels (Halls 1 - 8)	○	○	○	○	○
Student Hostels(Newer Hostels - Halls A - F)	○	○	○	○	○
Student Hangout Spaces	○	○	○	○	○
Fabrication Lab	○	○	○	○	○



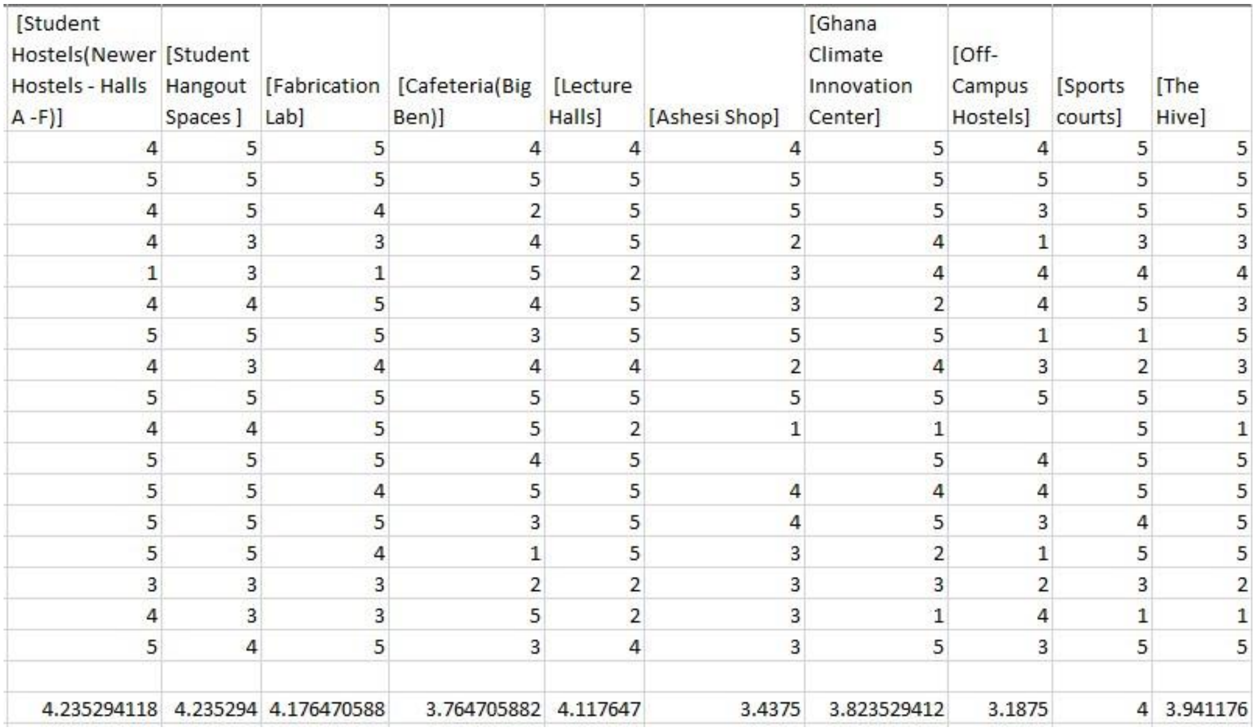
Cafeteria(Big Ben)	○	○	○	○	○
Lecture Halls	○	○	○	○	○
Ashesi Shop	○	○	○	○	○
Ghana Climate Innovation Center	○	○	○	○	○
Off-Campus Hostels	○	○	○	○	○
Sports courts	○	○	○	○	○
The Hive	○	○	○	○	○

Any comments that could enhance a touring experience in Ashesi University?

A.2 Questionnaire results



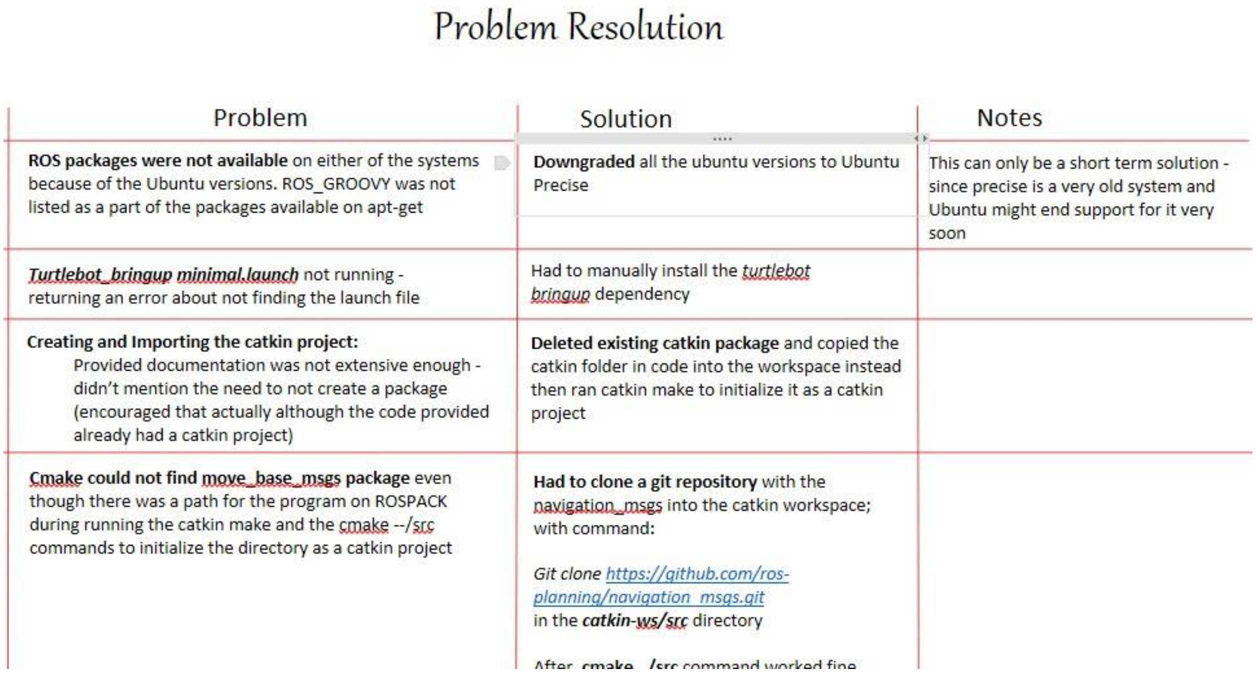
Library	[Faculty Offices]	[Computer Labs]	[Engineering Labs]	[Norton Motoulsky Hall]	[Cafeteria (Akonnor)]	[AshPitch]	[Health Center]	[Research Building]	[Student Hostels (Halls 1 - 8)]
4	4	4	4	4	4	5	4	4	4
5	5	5	5	5	5	5	5	5	5
5	1	1	3	5	5	4	5	5	3
5	3	3	4	3	3	3	5	3	5
2	2	2	4	4	5	5	5	5	1
5	4	5	5		3	3	5	5	4
5	4	5	5	5	4	5	5	4	5
4	3	4	5	4	4	5	5	4	4
5	5	3	5	5	4	5	5	5	5
3	1	5	5	4	4	5	5	5	4
2	2	2	5	2	5	5	5	5	
4	3	4	4	4	5	5	5	5	5
5	4	5	5	5	5	5	3	5	4
4	2	4	5	2	2	5	4	5	5
2	1	3	3	2	2	3	3	3	3
1	5	3	1	3	1	1	1	2	2
5	3	5	5	3	5	3	5	4	5
3.882353	3.058824	3.7058824	4.294117647	3.75	3.88235294	4.2352941	4.411765	4.352941	4



[Student Hostels(Newer Hostels - Halls A -F)]	[Student Hangout Spaces]	[Fabrication Lab]	[Cafeteria(Big Ben)]	[Lecture Halls]	[Ashesi Shop]	[Ghana Climate Innovation Center]	[Off-Campus Hostels]	[Sports courts]	[The Hive]
4	5	5	4	4	4	5	4	5	5
5	5	5	5	5	5	5	5	5	5
4	5	4	2	5	5	5	3	5	5
4	3	3	4	5	2	4	1	3	3
1	3	1	5	2	3	4	4	4	4
4	4	5	4	5	3	2	4	5	3
5	5	5	3	5	5	5	1	1	5
4	3	4	4	4	2	4	3	2	3
5	5	5	5	5	5	5	5	5	5
4	4	5	5	2	1	1		5	1
5	5	5	4	5		5	4	5	5
5	5	4	5	5	4	4	4	5	5
5	5	5	3	5	4	5	3	4	5
5	5	4	1	5	3	2	1	5	5
3	3	3	2	2	3	3	2	3	2
4	3	3	5	2	3	1	4	1	1
5	4	5	3	4	3	5	3	5	5
4.235294118	4.235294	4.176470588	3.764705882	4.117647	3.4375	3.823529412	3.1875	4	3.941176

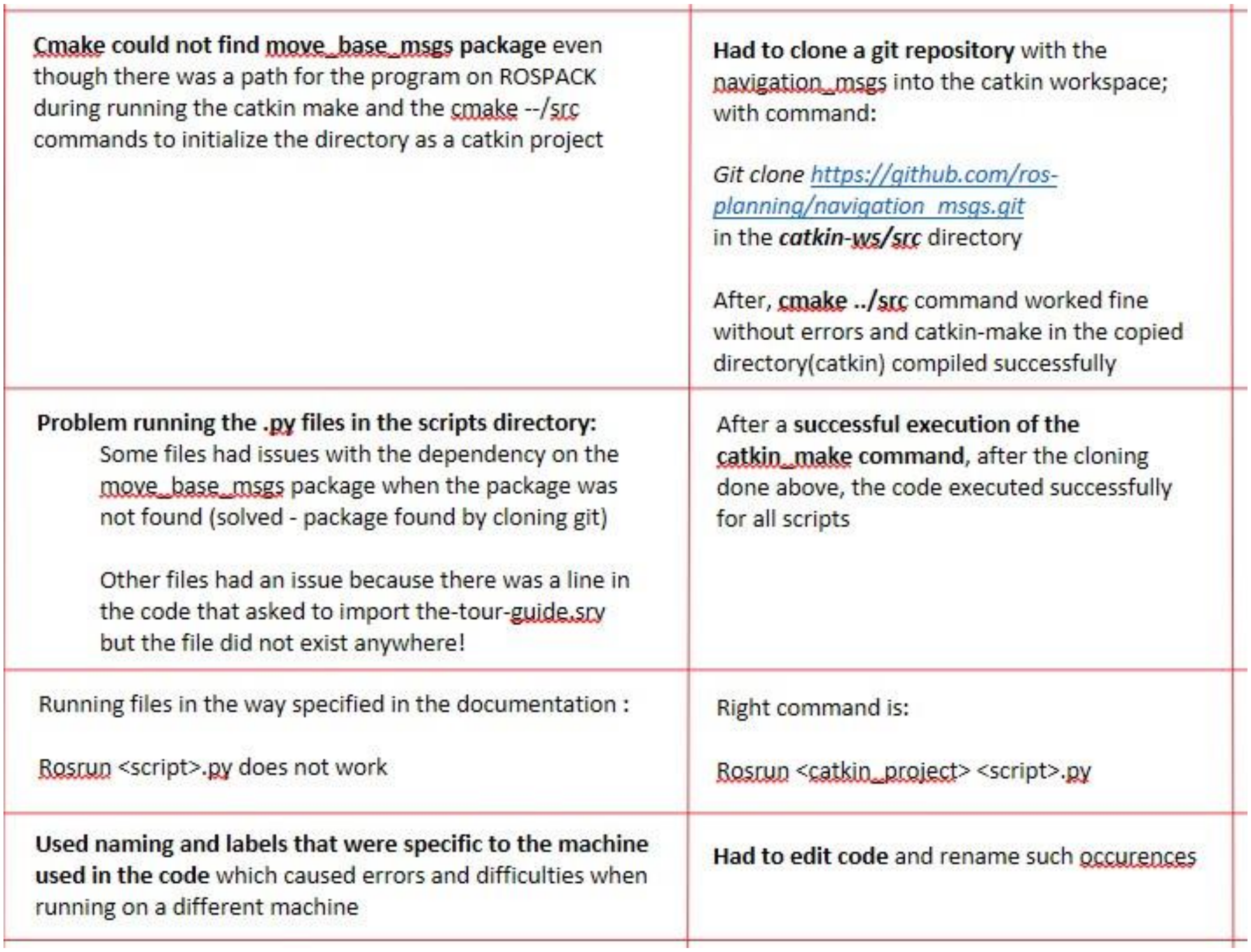
Appendix B[Implementation]

B.1 Problem-resolution log

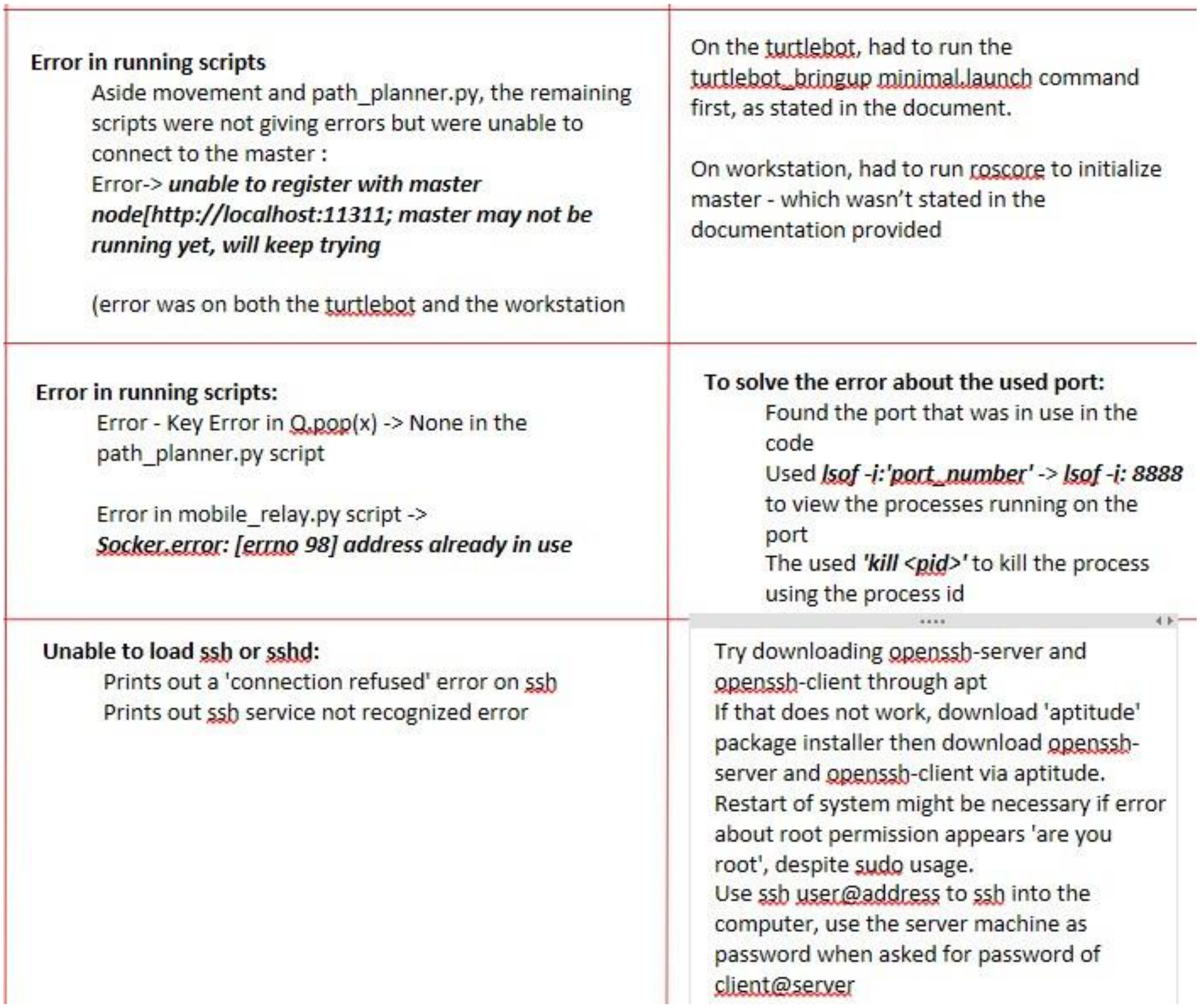


Problem Resolution

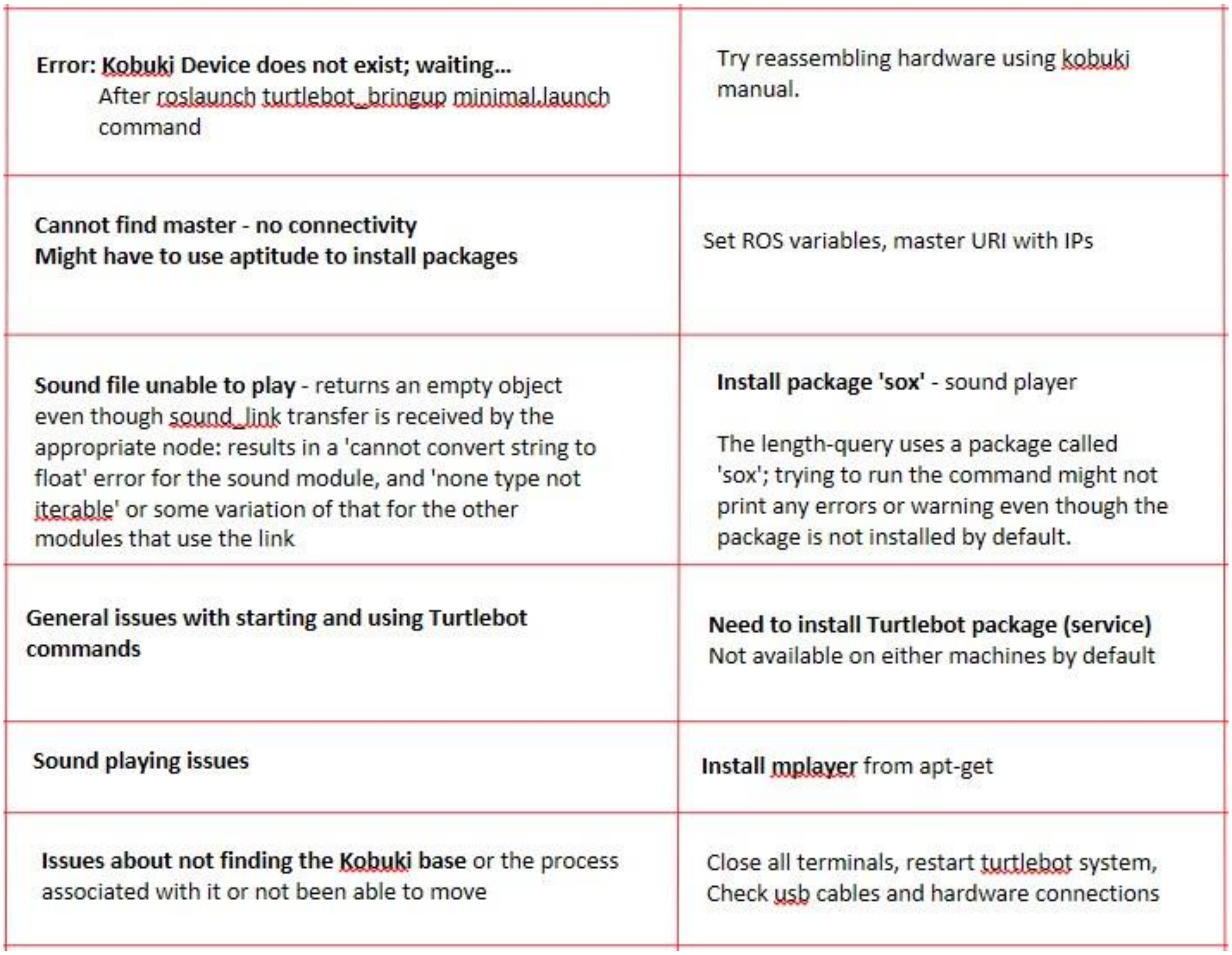
Problem	Solution	Notes
ROS packages were not available on either of the systems because of the Ubuntu versions. ROS_GROOVY was not listed as a part of the packages available on apt-get	**Downgraded** all the ubuntu versions to Ubuntu Precise	This can only be a short term solution - since precise is a very old system and Ubuntu might end support for it very soon
Turtlebot_bringup minimal.launch not running - returning an error about not finding the launch file	Had to manually install the *turtlebot bringup* dependency	
Creating and Importing the catkin project: Provided documentation was not extensive enough - didn't mention the need to not create a package (encouraged that actually although the code provided already had a catkin project)	**Deleted existing catkin package** and copied the catkin folder in code into the workspace instead then ran catkin make to initialize it as a catkin project	
Cmake could not find move_base_msgs package even though there was a path for the program on ROSPACK during running the catkin make and the cmake --/src commands to initialize the directory as a catkin project	**Had to clone a git repository** with the navigation_msgs into the catkin workspace; with command: *Git clone https://github.com/ros-planning/navigation_msgs.git* in the ***catkin-ws/src*** directory After **cmake** /src command worked fine	



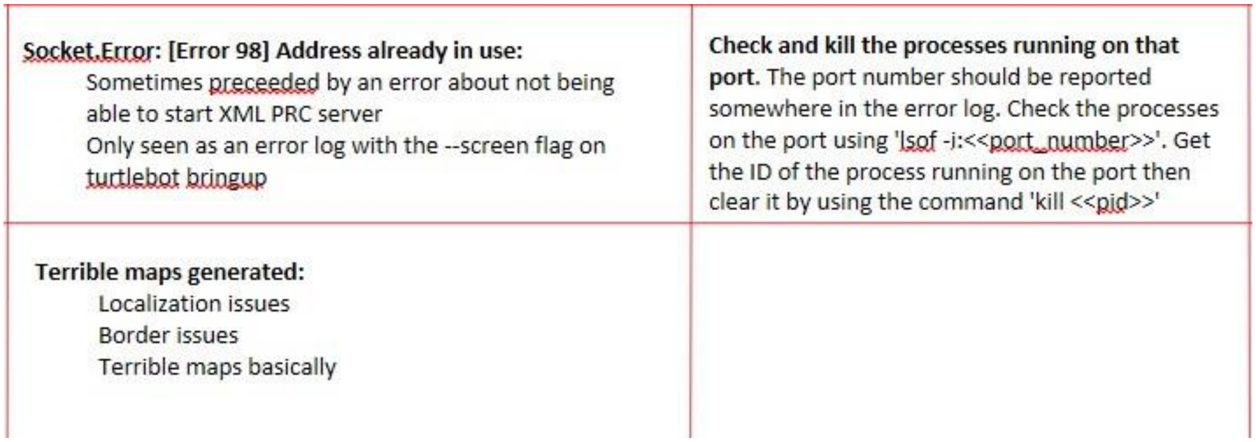
Cmake could not find move_base_msgs package even though there was a path for the program on ROSPACK during running the catkin make and the cmake --/src commands to initialize the directory as a catkin project	**Had to clone a git repository** with the navigation_msgs into the catkin workspace; with command: *Git clone https://github.com/ros-planning/navigation_msgs.git* in the ***catkin-ws/src*** directory After, **cmake ../src** command worked fine without errors and catkin-make in the copied directory(catkin) compiled successfully
Problem running the .py files in the scripts directory: Some files had issues with the dependency on the move_base_msgs package when the package was not found (solved - package found by cloning git) Other files had an issue because there was a line in the code that asked to import the-tour-guide.srv but the file did not exist anywhere!	After a **successful execution of the catkin_make command**, after the cloning done above, the code executed successfully for all scripts
Running files in the way specified in the documentation : Rosrun <script>.py does not work	Right command is: Rosrun <catkin_project> <script>.py
Used naming and labels that were specific to the machine used in the code which caused errors and difficulties when running on a different machine	**Had to edit code** and rename such occurences



Error in running scripts Aside movement and path_planner.py, the remaining scripts were not giving errors but were unable to connect to the master : Error-> ***unable to register with master node[http://localhost:11311; master may not be running yet, will keep trying*** (error was on both the turtlebot and the workstation	On the turtlebot, had to run the turtlebot_bringup minimal.launch command first, as stated in the document. On workstation, had to run roscore to initialize master - which wasn't stated in the documentation provided
Error in running scripts: Error - Key Error in Q.pop(x) -> None in the path_planner.py script Error in mobile_relay.py script -> ***Socker.error: [errno 98] address already in use***	**To solve the error about the used port:** Found the port that was in use in the code Used ***lsof -i:'port_number'*** -> ***lsof -i: 8888*** to view the processes running on the port The used ***'kill <pid>'*** to kill the process using the process id
Unable to load ssh or sshd: Prints out a 'connection refused' error on ssh Prints out ssh service not recognized error	Try downloading openssh-server and openssh-client through apt If that does not work, download 'aptitude' package installer then download openssh-server and openssh-client via aptitude. Restart of system might be necessary if error about root permission appears 'are you root', despite sudo usage. Use ssh user@address to ssh into the computer, use the server machine as password when asked for password of client@server



Error: Kobuki Device does not exist; waiting… After roslaunch turtlebot_bringup minimal.launch command	Try reassembling hardware using kobuki manual.
Cannot find master - no connectivity **Might have to use aptitude to install packages**	Set ROS variables, master URI with IPs
Sound file unable to play - returns an empty object even though sound_link transfer is received by the appropriate node: results in a 'cannot convert string to float' error for the sound module, and 'none type not iterable' or some variation of that for the other modules that use the link	**Install package 'sox'** - sound player The length-query uses a package called 'sox'; trying to run the command might not print any errors or warning even though the package is not installed by default.
General issues with starting and using Turtlebot commands	**Need to install Turtlebot package (service)** Not available on either machines by default
Sound playing issues	**Install mplayer** from apt-get
Issues about not finding the Kobuki base or the process associated with it or not been able to move	Close all terminals, restart turtlebot system, Check usb cables and hardware connections



Socket.Error: [Error 98] Address already in use: Sometimes preceeded by an error about not being able to start XML PRC server Only seen as an error log with the --screen flag on turtlebot bringup	**Check and kill the processes running on that port.** The port number should be reported somewhere in the error log. Check the processes on the port using 'lsof -i:<<port_number>>'. Get the ID of the process running on the port then clear it by using the command 'kill <<pid>>'
Terrible maps generated: Localization issues Border issues Terrible maps basically	