The ‘Guide Robot’ Project Report

Authors:
Tjark Siewers, Matthew Yates, Aleksandr Kostrub,
Negrita Kalcheva, Serban Bogdan, Vivek Chauhan

The University of Birmingham. Intelligent Robotics.
Birmingham, United Kingdom

This report describes and documents a robotic system capable of self-localization, path finding, obstacle avoidance, speech recognition and production with all the underlying mechanisms to perform successful human guiding in the world with the premade digital spatial map. The aim of building this robot is to create a robust, modernized successor to older guide robots such as MINERVA [4] with the use of newer algorithms and the implementation of voice control. The project was carried out as the part of the course fulfillment.

1. INTRODUCTION

Intelligent robotics is an interdisciplinary branch that combines both science and engineering to research and create robotic systems capable of imitating human intelligence and/or learning. It is common knowledge that since the middle of the 20-th century, robots are used in various industries, such as manufacturing, military, assembly, research, education, and even adult entertainment. However, it is not a widely spread knowledge, that first mentions of robotic systems date back as far as several centuries B.C., from various locations around the ancient world, but mainly from the East [1]. Early mentions of robotic systems feature human-like robots, bird-like robots, and systems that could represent devices from various stages of industrial revolutions – fire engine, a wind organ, and a steam engine. It is uncanny that for centuries humanity attempted to achieve success in making of complex and intelligent systems that would benefit their life.

For a successful intelligent operation of a robot with and within the material world, it would require a way to make sense of it as any other intelligent system in it. From single cellular organisms to human beings, everything and everyone has a way to sense its environment as a vital prerequisite for survival.

Since the early end of the last century, digital advances in object programming [2] allowed the humanity to advance their millennial attempts at creating such systems. Currently, robotics is within reach of its booming point, however, many topics still require more elegant solutions. One of such topics is Simultaneous Localization and Mapping (SLAM). SLAM allows robots to traverse through space in the material world using sensors and algorithms in path planning as well as building or updating a map of the surroundings [3]. This is the single most essential part of any mobile robot that acts as a base for building additional functionality as it, literally, makes a robot mobile. For that reason, SLAM has been a very popular computational problem in intelligent mobile robotics for the last half of the century. This project was built around this problem to create an elegant solution allowing confident future development.

2. BACKGROUND

Task Planning

The team decided to make a guide robot with speech recognition/production capabilities. To achieve this goal, a great deal of effort went into the planning of the theoretical prerequisites for such an execution.

There were many robots in the last 75 years that focused on SLAM and/or obstacle avoidance, starting from tortoise robots of William Grey Walter that mimicked insect behaviour, ‘Shakey’ – the first general-purpose mobile robot, and Boston Dynamics robots that focused on robot’s mobility in the world.

Currently, the guide robots are mostly used in the museums [4] [5], where they can give a tour to everyone who is not able to attend the distant location, or at nights, when the museum is closed and empty. However, there is a bright future for guide robots, ‘LG’ promises to use guide robots in airports that would be able to direct passengers to their gates, pick up trash, and hoover [6].

The fact that the guide robot operates in a limited world is a benefit to focus on developing precision, behaviour, and localization that can serve as a stepping stone for developing a guide robot in the wild world. Eventually, robots will leave research laboratories and university basements to be embraced by the world in day to day tasks as general-purpose helpers. Before that day comes, it is vital to make sure that they are safe and reliable. The team has undertaken a duty to help create such a safe and reliable guide robot.

An example of the task – the robot would be available at the certain location of the floor, where it would be asked to guide the person to a certain landmark in the building. The robot then would depart to such landmark, avoiding obstacles and calculating the most efficient route from starting point to the landmark. Once the landmark was reached, the robot would inform the person and after a short delay begin moving to the starting location where it would be ready to accept the next task.

The required parts for an execution would be, apart from essential components (ROS, Robot, Ubuntu, Python), a set of planners (global, local) to plan the path from and to the selected location, localization algorithm for the robot to be able to relate its current location to the material world, a map of the available world, speech recognition to interpret commands given via voice, and speech production to inform the user about important events during the process of guiding. Last, but not the least, the robot would have to be safe both for itself and the environment.

To ensure the robot would not damage itself or anyone, several tests and parameter tweaking would need to take place.

Bayes Theorem

Bayes theorem is the equation which is used to update the probability of the where the robot is based on its sensor readings. It is a conditional probability model (probability of the event A, given the event B occurred). This is used to modify the robot's beliefs about its location when it receives new information on its probable location. This is represented as a probability between 0 and 1. Whenever the robot receives this new information it will incorporate this into its existing model to give a new and more accurate estimation.

Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization (AMCL) is a particle filtering technique which incorporates both Bayes theorem and Markov chains. This is used to localize the robot when it is given a map of the environment. This is an effective method of localization as it accounts for noise in its sensor readings of the environment into the model.

AMCL works by creating a cloud of particles on the map which are scattered randomly across the entire area. The robot then takes in an initial belief of where it may be, using each particle as a potential location. Based on the information gained from the robot's sensor model, a weight is assigned to each of the particles based on the likelihood that the robot's actual position is at that location. The weights are then normalized into a probability distribution. Based on this probability distribution the particles are then resampled with their new positions updated to reflect the robot's new beliefs about its location. This process then repeats itself with new information until the particles converge to a singular area which should be a representative of the robot's real-world position. During the resampling phase, the number of particles distributed varies according to the robot's certainty of its position, this allows for a more efficient use of computational resources.

Speech recognition

Speech recognition transfers speech to data by converting the sound waves into a digital signal through the process of sampling. This digital representation of the sound is a measurement of the sound wave at small but discrete time intervals. During this process, background noise is filtered out and the sound wave may be split up depending on frequency to differentiate between changes in pitch in the input voice.

The digitalized soundwaves are then split up according to which phoneme they closest represent. Phonemes are small segments which make up words in the spoken language (there are 44 of them in the English language). The program then uses this information to put together words and sentences. Modern implementations of speech recognition (e.g. Google) use complex recurrent neural network models to infer the context of the input in order to give a better prediction on what the spoken phrase was.

3. DESIGN

3.1 System components

3.1.1 Hardware components

This study used a variety of hardware components in order to complete the given task.

Pioneer P3-DX

The main hardware component was a Pioneer P3-DX mobile robot. It is a lightweight two-wheel two-motor differential-drive robot equipped with a dedicated motion controller with encoder feedback. The robot is equipped with a platform situated at 237 mm above the ground. The platform is 381 mm wide at its narrowest point and 455 mm long. Also, situated at 210 mm above ground level, the robot is equipped by default with 8 forward-facing ultrasonic (sonar) sensors. The robot can develop a maximum forward speed of 1.2 m/s and a maximum rotational speed of 300 degrees/s. The robot serves as the main platform on which other peripherals are attached to.

Hokuyo URG-04LX

Hokuyo URG-04LX is a laser rangefinder and it is used for gathering precise information about the environment the robot is in. The sensor has a minimum range of 20 mm and can get readings at a maximum range of 4000 mm. It has an area scanning range of 240° with an angular resolution 0.36° (the angle between two consecutive laser beams), meaning a total of 683 beams. The diameter of the laser beam depends on the distance, having a diameter of 20 mm at a distance of 2000 mm and a diameter of 40mm at the maximum range (4000 mm). The sensor is mounted on top of the robot platform, in the centre of the frontal side of it at a height of 300 mm from the ground level.

Orbecc Astra Pro

Orbecc Astra Pro is a depth camera which provides superior depth resolution and reduced latency. It includes a 720p 2D colour camera which outputs video at 30 frames per second and a 3D microchip used for depth images. The depth cameras can detect objects that are situated at a minimum distance of 0.6 m and a maximum distance of 8.0 m. The camera has a horizontal field of view of 60° and a vertical one of 49.5°, resulting in a diagonal field of view of 73°. The camera is equipped with two microphones and it is connected to the laptop through USB. The camera is mounted on a tripod fixed on top of the robot platform, at a height of 150 cm above the ground level. The camera is tilted at an angle of -**10** ° (where 0° means the camera is parallel to the ground).

Control Unit

The control unit of the robot was a Lenovo E31-70 laptop equipped with an Intel Core i3-5005U dual-core CPU operating at 2000 MHz and 4096 GB of DDR3 single-channel RAM operating at 1600 MHz.

3.1.2 Software

Ubuntu

Ubuntu is an open-source operating system. It is a Linux distribution based on the Debian architecture usually run on personal computers, but also popular on network servers. For implementing the solution, Ubuntu 14.04 LTS was used.

Robot Operating System

Robot Operating System (ROS) is a set of frameworks designed for robot software development released in 2007, which provides services as message-passing between processes, low-level device control, package management, hardware abstraction (allows the writing of device-independent code). Given the fact that most of the libraries have a rich collection of dependencies on open-source software, ROS is usually run on Linux – based systems, variants for Windows are considered as being 'Experimental'.
The main unit of ROS is the **node.** Instead of running a large complex program to provide control to the robot, the framework supports writing smaller, more task-oriented programs. These nodes communicate with each other (pass data) using **messages**, which are published on **topics**. A topic is a separate communication channel where nodes can publish messages having a specific format. In order to receive messages, nodes subscribe to topics.
ROS processes are represented using a graph – like architecture, where nodes represent tasks and arches are channels (topics) through which messages are transmitted between processes.

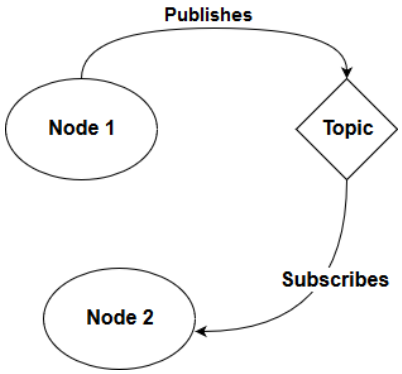

Figure 1 Basic ROS node interaction

Related ROS nodes are grouped in packages that can be easily compiled and run. A various range of packages is required in order to build a ROS robot control system.

3.2 Localization

Localization is a key factor to the key problem of "Where am I?" which is a fundamental problem in the field of mobile robots. It is used to provide an estimated pose of the robot in the world, at a certain point in time based on sensor readings (laser scans).
In building the solution, a variant of AMCL provided in ROS navigation stack was used, over a bespoke AMCL implementation previously developed. The decision of using one variant over another was based on a series of experiments that have been conducted (described in sections **4.1** and **4.2**), resulting in the fact that the ROS variant of AMCL performs better and provides a much higher accuracy of pose estimation, and a higher confidence level, therefore resulting in a robust localisation that is reliable in a natural environment. Also, the ROS implementation of AMCL provides the option of tuning by changing values to a predefined set of parameters in order to increase the performance of the algorithm in the given environment.
Based on the results of a series of runs with different parameter settings, the package was tuned to use a minimum of 100 particles (when the confidence of the localization was high) and a maximum of 5000 particles (in the case of low confidence or delocalization). Also, an expected side deviation of 10 cm for forward translation (measured on a distance of 2.5 m) was recorded and fed as a parameter. The values of the rest of the parameters were used with their default values.

3.3 Path Planning

Path planning was using the Probabilistic Road Map technique. ROS Navigation Stack, containing the AMCL package previously described also provides a solution to the path planning problem: a package named **move_base**. This package provides an interface which allows the developer to configure, run and interact with the navigation stack on a robot. The main node of this package interacts both with components of the navigation stack and also other components of the robot in order to be able to try and reach a goal set in the world by planning a path to it, sending movement commands to the robot and use sensors to avoid obstacles.
The **move_base** package can be considered as consisting of a set of low – level components that interact with each other in order to be able to plan a path to a goal and move the robot base along the path towards the goal. The package uses two planners, one local and one global, and two costmaps, a local and a global one and a set of recovery behaviours.

Costmaps

The costmaps use sensor readings to mark (add obstacles on the maps) or unmark (remove obstacles from the map) based on the movement of the robot. Each cell of the costmap can be marked with a value between 0 and 255, and based on the values a cell can be either occupied, free or unknown. Each cell marked as

an obstacle gets inflated by the inscribed radius of the robot. This allows the planning of routes such as the footprint of the robot should never intersect a cell marked as an obstacle if the centre point of the robot will not cross the limit of the inflated cloud.
ROS Navigation Stack heavily relies on two costmaps, a global costmap, which is generating by inflating the obstacles provided to the navigation stack through a static map, usually using map_server, and a local costmap, which is generated in real time by inflating the obstacles detected by the sensors mounted on the robot.
In the implementation of the project, the costmaps were configured with an inflation radius of 0.5 m and used as observation sources for marking obstacles both the laser range finder and the depth camera. Obstacle clearing from the costmap was performed using only the laser rangefinder.
The size of the local costmap was 6.5 x 6.5 m, and it was updated at a frequency of 5.0 Hz. The update rate of the global costmap was 2.0 Hrz. Thre rest of the parameters were used with the default values.

Global Planner

The global planner is the component of ROS navigation stack that provides a function that given a goal in the world, can provide the robot mobile base with a plan. This planner uses the costmaps to compute a minimum cost plan from the current location of the robot to the goal that was set. ROS stack provides three variants of global planners: Global Planner, Navfn, and Carrot Planner. After conducting an experiment which consisted of a comparison between Global Planner and Navfn, it was concluded that Navfn will better suit the requirements of the project that was developed. Navfn is a global planner designed for circular robots that uses Dijsktra's algorithm to compute the minimum path to a set goal. The decision of using this algorithm over Global Planner was also supported by the fact that Navfn allows planning close to a specified goal if an obstacle is recorded to be in the same location as the goal, while Global Planner fails in providing a plan in this scenario. Navfn was tuned to use a default tolerance of 100mm, meaning that in the scenario that there is an obstacle in the location where a goal was set, then the planner will try to plan a new goal as close as possible to the initial goal, but not further than 100 mm. The rest of the parameters were used with the default values.

Base Local Planner

Base Local Planner is a package of ROS Navigation Stack that is responsible for the actual movement of the robot in the plane. Its purpose is to act as an interface between the robot and the global planner, meaning that using the global plan provided by the global planner, the local planner is required to create a value function that computes the cost values of different trajectories that the robot could use to move and choose the one which has the lowest cost. Using this trajectory, local planner sends forward and rotational velocities to the robot. In the project, TrajectoryPlannerROS was used over the TebLocalPlanner due to limited computational power, the latter requiring more resources in order to perform efficiently.
TrajectoryPlannerROS provides a large set of parameters that allow the developers to fine tune the local planner in order to get the maximum performance in the world. The tuning of the local planner was performed through repeated runs where the robot had to move towards a specified goal and its behaviour was studied. First, robot parameters were set, according to the specifications of the Pioneer P3-DX robot used:

- forward acceleration limit of 0.5 m/s^2;
- rotational acceleration limit of 1.0 $radians/s^2$;
- the maximum forward velocity of 0.9 m/s;
- the minimum forward velocity of 0.1 m/s;
- the maximum rotational speed of 1.6 radians/s;
- the minimum in-place rotational speed of 0.6 radians/s

A forward simulation time of 3 seconds was used to simulate trajectories obtained by sampling 6 forward translation speeds and 20 rotational speeds. The robot forward simulation distance was set to 0.65 m.
Cost function parameters were tuned to improve the capability of the robot to reach its' goal:

- a value of 1.2 to the weighting for how much the controller should attempt to stay close to the path (**pdist_scale**);
- a value of 1.0 to the weighting for how much the controller should attempt to reach its local goal (**gdist_scale**);
- a value of 0.02 to the weighting for how much the controller should attempt to avoid obstacles.

The rest of the parameters of TrajectoryPlannerROS were used with their default values.

Recovery behaviours

ROS Navigation Stack has a set of built-in recovery behaviours designed to get the robot to unstuck itself in the scenario that he gets stuck on its' way towards the goal.
In the implementation of the project the planner was configured to use two types of recovery behaviours: **clear costmap**s, which means reverting the local costmap to the state of the global one, and **rotate recovery**, which means that the robot will stop and do a 360° rotation in place. The rotation behaviours are run in a sequence, starting with clearing the costmaps and continuing with rotate recovery. A 5 seconds controller patience was used to make the recovery behaviours run after a delay of 5 seconds (when the robot gets aware that it is stuck, waits for that amount of time then it starts the recovery behaviours).

3.4 Speech Recognition

Speech recognition is a field of computational linguistics that builds methodologies which allow computers to recognize and translate spoken language into text. In the building of the project, speech recognition was used in order to send commands to the robot.

Given the fact that the project required high accuracy in recognizing spoken commands, Google Cloud Speech API was used over the local implementation of Pocketsphinx. Google Cloud Speech API uses powerful cloud-based neural networks to perform the conversion of audio into text, providing an easy-to-use framework. Speech recognition package for Python was used in order to easily configure and use the API. The package provides an interface that allows developers to continuously use the microphone in order to record audio phrases of predefined length and use Google Cloud Speech API in order to get the corresponding text translation of the audio file. Audio commands were recorded using the microphones from Orbecc Astra Pro camera mounted at a height of 150 cm above the ground, so it is as close as possible to the person giving the command. Before starting each listening session, the microphone automatically adjusts its sensitivity based on the ambient noise, so it reduces the background noise for each audio record it sends to the API. A set of preferred phrases containing the names of each possible destination on the map were provided to the API in order to increase the probability of outputting those values. Also, a maximum length of 10 seconds was set for each recording session.

3.5 Text to Speech

The system was required to provide status updates to the user as it was running. For achieving that, a text to speech module was used in order to produce dynamic speech synthesis. In the implementation, the **pytts** package was used for speech reproduction. It is based on FreeTTS Linux speech synthesis. Speech reproduction is done by using a function provided in the package which receives as an input a text parameter and uses the sound output device to output the corresponding speech. In order to avoid concurrent requests, each call was added into a queue.

3.5 The tour algorithm

The tour algorithm uses all the other components described in order to perform a tour of the world he is in. The algorithm could be described using the following:

1. build a queue of goals correspond to a landmark on the map that the robot is aware of;
2. remove a goal from the queue and plan a route towards it;
3. move to the goal;
4. when the location is reached check the status of the queue;
5. if the queue is not empty, then go to step 2;
6. if the queue is empty, then move to the waiting location.

3.6 Control system

The control system was implemented as a finite set of states where the robot can be only in one state at a time.

The robot starts by receiving an estimate of his pose in the world. Checking the confidence of the given pose, the first step

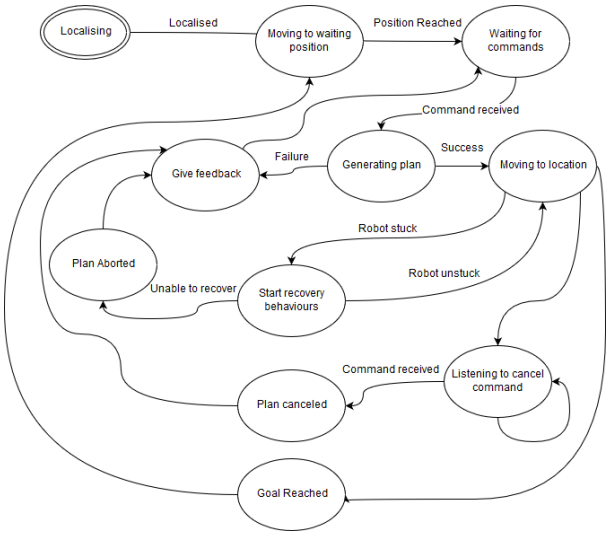

Figure 2 Control system states

in the behaviour is to start localizing itself by moving along a wall until it gathers enough information to be confident about his position. When the confidence of the pose estimate is over a predefined threshold for more than 5 seconds, the robot moves toward the pre-set waiting position. If the waiting position is reached, the robot adjusts the sensitivity of the microphone for 5 seconds in order to reduce the background noise, then it goes into the 'awaiting command' state when it awaits for a voice command that it recognizes in order to perform an action. If a valid command is received, the path planning is performed. If a valid plan cannot be found, then feedback is provided to the user and the robot continues waiting for new commands. If a valid plan is found, then the robot begins heading towards the goal. During this time, it continuously listens to voice commands, waiting to receive a cancel command. If the goal is canceled, feedback is provided to the user and the robot is heading back to the waiting location. During its movement towards the goal, the robot might find that obstacles are recorded along the path and may be blocking the way. After trying to plan an alternate route, if none is found, then recovery behaviours are started, one after another, trying to clear the obstacles off the costmap. If after running all the recovery behaviours the robot is still stuck, then the plan is aborted, feedback is provided to the user and after a short delay, the robot heads towards the waiting location. If the path gets cleared, the robot continues heading towards the destination. When the goal is reached, the user is provided with feedback and after a short delay, the robot heads back to the waiting location.

4. EXPERIMENTATION

4.1 Confidence comparison for different AMCL Implementations

4.1.1 Aim

The aim of this experiment is to compare our own AMCL implementation and the AMCL implementation provided by ROS. The measure being tested is the system's confidence of its localization.

4.1.2 Method

In this experiment, we measured the confidence of the localization over time. The confidence is calculated by taking the inverse of the variance of the particles in the particle filter incremented by one. The closer the particles are, the smaller the variance and therefore the larger the inverse becomes. Since the minimum value of the variance is 1 (0+1 = 1), the maximum confidence value is 1/1 = 1. This way of calculating confidence maps it to the values between 0 and 1. The formula for calculating confidence is: confidence = 1 / (variance + 1).

To make the experiment fair, we ran the robot using our AMCL and ROS AMCL on the same path on the lower ground floor map. Three runs were performed with each variant of AMCL and the final results were averaged (mean).

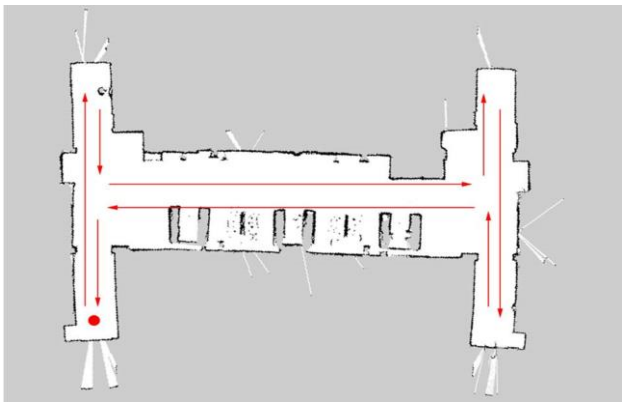

Figure 3 Path followed for conducting the experiment

4.1.3 Results

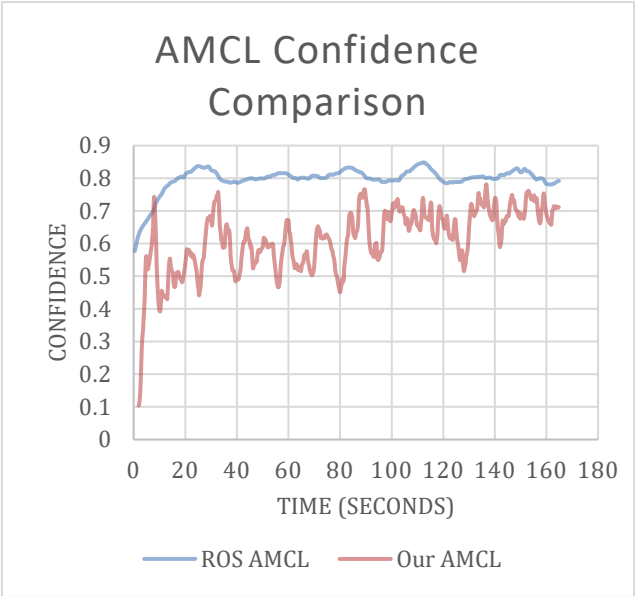

Figure 4 Comparison of confidence levels for different AMCL implementations

From the graph, we can see that the ROS AMCL implementation on average provides higher confidence. Changes in confidence are smoother than for our AMCL implementation.

4.2 Localisation Accuracy of different AMCL

4.2.1 Aim

The aim of this experiment was to compare adaptive Monte Carlo localization provided by ROS navigation stack and the implementation created by the team to determine the accuracy of the robot localization.

4.2.2 Methods

The parameter configurations of both algorithms that were tested changed the amount of noise, the maximum number of particles and the minimum number of particles.

The readings were taken at 4 different checkpoints as shown in **Figure 5** on a predefined map of lower ground floor area. The experiment was conducted from an initial position of a robot for both localization algorithms. The robot was instructed to move to next position where estimated position was recorded; this process was repeated until the robot visits all the points to collect all the necessary data. The team ran this experiment 3 times for both AMCL implementations to check the robustness of the algorithms.

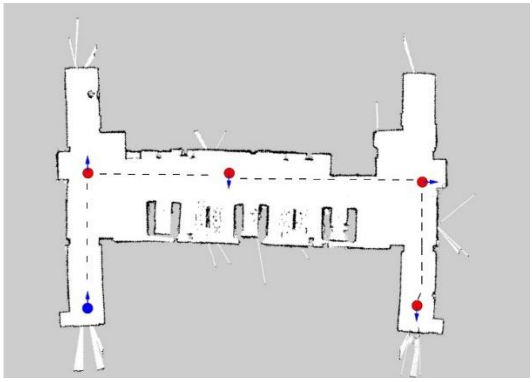

Figure 5 AMCL accuracy experiment path and measurement locations

4.2.3 Performance:

The performance of each localization algorithm was determined by the difference of the actual and estimated position and to compare both algorithms, the records were gathered simultaneously.

In this experiment, the following parameters were used:

Algorithm Name	Parameters		
	Noise (m)	Min Particles	Max Particles
ROS AMCL	0.02	100	5000
Our AMCL	0.02	200	1000

Figure 6 AMCL parameters settings

It should be noted that the term 'Noise' is referring to the maximum side deviation expected from the forward translation of the robot (drift).

Steps to perform the experiment:

- Start from the initial position of the robot in the simulator.
- Move to point 1
    - Record x and y coordinates of the robot at the estimated pose location
    - Record x and y coordinates of the robot in stage
    - Calculate the difference between the actual location and estimated location
- Repeat until robot visits all the points.

4.2.4 Results

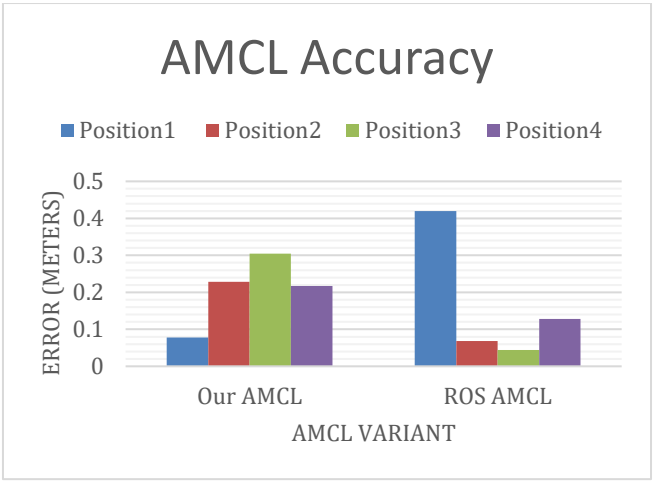

Figure 7 AMCL Accuracy results

The results clearly show that the AMCL implemented by ROS had fewer errors in comparison to our AMCL algorithm. According to the **Figure 7**, at the first location, ROS AMCL had a significant amount of error around 0.41 meters while our AMCL had the least error around 0.07 meters. The overall error of ROS AMCL and our AMCL was around 0.16 and 0.20 meters respectively.

4.2. Comparison of global planner performance

4.3.1 Aim

The aim of this experiment is to compare the performance of 3 different global planners to find which one was the most efficient. The global planners evaluated were – the navfn provided directly from the navfn ROS library, a global planner based on the Dijkstra search algorithm and a global planner based on the A* search algorithm.

4.3.2 Methods

The vehicle was driven along a set path. For each global planner variant, 4 runs were completed using the same course for each. To evaluate the performance in a crowded environment, several obstacles were placed in the path of the robot, again these were kept the same for each run. The course used can be seen in F**igure 8.**

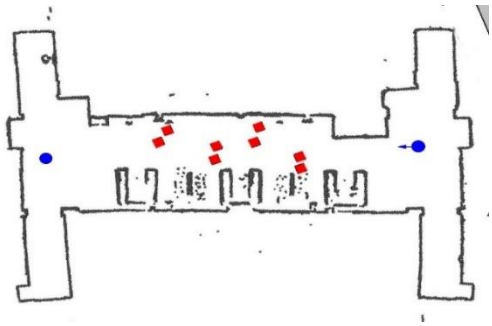

Figure 8 Obstacle course for the experiment

Figure 8 shows the course that the robot took. The right blue circle indicates the starting position and the left one is the end point. The red squares represent obstacles the vehicle had to navigate around.

4.2.3 Results

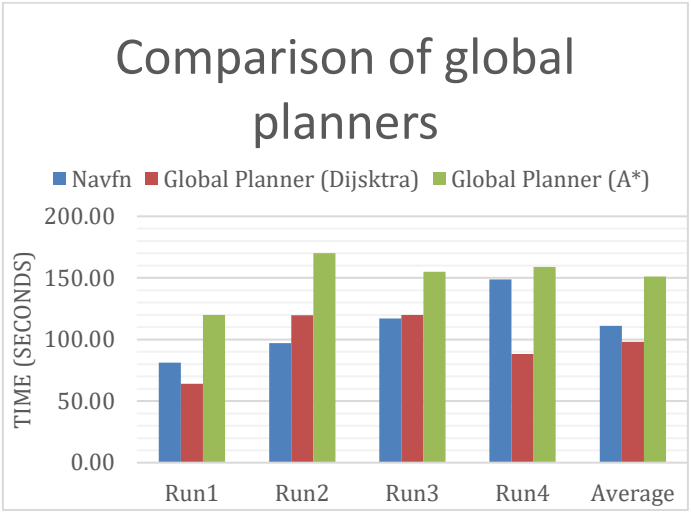

Figure 9 Global Planners comparison

On the horizontal axis in **Figure 9,** the four separate runs of the different planners are instead grouped into four runs containing three runs each for each global planner used. This is done to make the comparison easier to make and for it to be more detailed. In addition, the average time for traversing the route for each planner is shown. On the vertical axis of the bar chart in **Figure 9,** the time (in seconds) it took to complete an obstacle course using each global planner is shown. The four runs conducted using the navfn global planner are represented in blue bars in **Figure 9**. Respectively in orange, the runs using the global planner implementing the Dijkstra algorithm are shown and in grey, the runs using the global planner implementing the A* algorithm are shown in green. For the first run, as can be observed in **Figure 9,** the Dijkstra-based planner significantly outperforms both of the other planners from the tests. This is evident in the fourth run, too. Navfn's performance is close to Dijkstra's in the third run but it outperforms Dijkstra a little bit. Navfn outperforms Dijkstra more significantly in the second run. The A*-based global planner uses more computational resources than the other planners and performs the worst in all four cases.

Overall, it is visible in **Figure 9** that the navfn global planner and the Dijkstra - based global planner perform better than the A*-based global planner, at least the conducted tests on this specifically chosen route lead to this conclusion. This route was selected to be as sophisticated as the environment permits - many obstacles were positioned in the path of the robot, each of which the vehicle had to find a way to avoid and then plan its way ahead. Although navfn performs best in most of the cases it is outperformed in some of them by the Dijkstra-based global planner. That can be seen in the bars measuring the average time in seconds to traverse the path for each planner in **Figure 9**. From this, it can be summarised that the Dijkstra – based global planner performs best on average.

4.4. Comparison of local planner parameters

4.4.1. Aim:

The aim of this experiment was to compare the performance differences to the local planner when different parameters were used, in order to find the optimal values.

4.4.2. Methods:

The readings were taken at a target point in the predefined lower ground floor area. The experiment was conducted using the obstacle course shown in **Figure 10**. The starting position of the robot is marked with the blue circle and an arrow. The robot was then instructed to move towards the target position (blue circle on the left) while avoiding obstacles. At the target location, we recorded the time which the robot took to perform this task. The team ran this experiment 3 times for each set of parameter configuration to determine the one that provides the best performance.

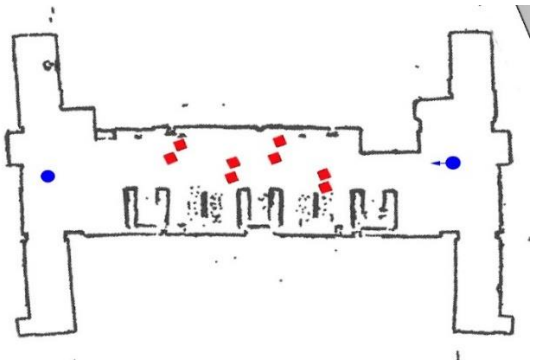

Figure 10 Obstacle course for local planner experiment

4.4.3. Performance:

The performance of this task was determined by the difference of time taken by the robot.

Based on previous observations, the team has decided to experiment with different combinations of values for two

parameters of the local planner, parameters which have proven to be the most influential on the performance of the local planner. The parameters used were: **sim_time**, which represents the amount of time spent by the planner to forward-simulate each trajectory (measured in seconds) and **heading_lookahead,** which represents the length of each simulated trajectory that the algorithm will compute the costs.

	Simulation Time	Heading lookahead
Config 1	3s	0.65m
Config 2	4s	0.4m
Config 3	4s	0.8m

Figure 11 Parameter configurations

Steps to perform the experiment:

- Start from the initial position of the robot in the simulator;
- Set a goal in the target point;
- Record the time for completing the course.

4.4.4. Results:

The results clearly show that the performance of the base local planner was the highest when we used the second configuration because the robot took the average of 62.33 seconds to reach the destination. On the other hand, the robot took almost twice time for other two configurations with around 130 and 140 seconds for first and third configuration respectively.

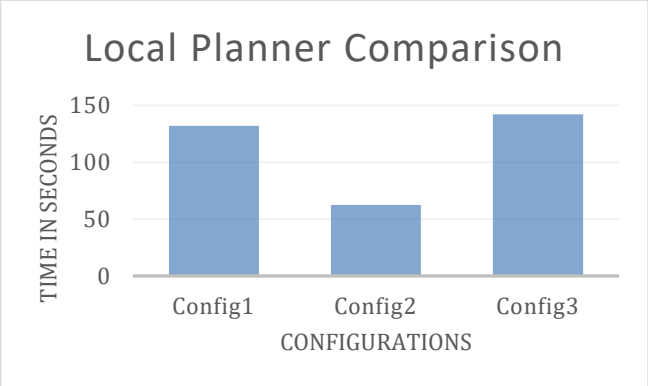

Figure 12 Average time for completing the course

5. DISCUSSION

After running the tests on the guide robot, the team concluded that the robot could perform reliably and confidently with given configuration in the observed world environment during busy times, successfully and efficiently avoiding obstacles and guiding users to the desired location. Given the feature of autonomous charging, the robot could function constantly to provide information, directions, and amusement to the public of the selected environment.

In comparison to previous century mobile robots, the robot has vast memory capabilities, internet connectivity, and is not limited to basic functionality. In its current state, the robotic system can serve as a strong building base for additional modules and constructs. Since the aim of the research was to explore and create a reliable and accurate robotic guiding system for features to be built on, the team acknowledges that it has achieved the goal.

5.1. Improvements

Although our robot was robustly built and worked well at its set tasks in practice there are a number of improvements which could be made to improve its functionality, performance, and efficiency. One improvement to make would be to use a better microphone for the robot. A higher quality microphone would be useful as the one used was not very sensitive and often required the person to stand very close to it for it to work. A better microphone would be able to detect commands and ignore background noise more easily, improving its performance in crowded environments.

Aside from hardware changes, one improvement which could be made would be to add more phrases to the range of input commands it can recognize. This would mean the phrases used to command the robot would have to be less specific. Another potential improvement to its voice functionality could be to use a voice recognition program which is able to be used offline, as the current implementation uses Google's speech API and therefore can experience latency in its response. While it would be more computationally expensive to use an offline program, it could be more beneficial to this specific robot as it only needs to know a few key phrases.

Face recognition could be implemented so the robot could actively approach and engage with people rather than the current version which requires a person to initiate it. This could also be useful for providing a log of different people it interacted with. Like the previous suggestion with the voice control, this improvement would be more computationally expensive to run than the current design which means it might need more powerful hardware for it to run smoothly.

Lastly, the main improvement would be to develop a similar functionality of the guide robot in the open world where the robot could assist and travel efficiently and safely to aid the general public and possibly disabled population. This would not limit the operation of robots to airports or museums. This idea could effectively fill a gap between self-driving cars and closed world guide robots.

5.2 Conclusion

While the design of the robot was not very complex the individual modules of the robot were well integrated together, leading to a robust performance by the system as a whole. The robot managed to effectively localize, navigate to a destination point, avoid obstacles and take commands from speech input. The performance of the robot was also improved through the process of experimentation in both simulation and reality. Experimentation was particularly useful in deciding which algorithms to use for navigation for both the local planner and the global planner. While the robot did work, there are improvements to be made and the design has room for added complexity. In conclusion, the achieved robot state has fulfilled the project outline of a guide robot capable of voice operation.

6.REFERENCES:

[1] Needham, Joseph (1991). Science and Civilisation in China: Volume 2, History of Scientific Thought. Cambridge University Press. ISBN 0-521-05800-7.

[2] R. J. Popplestone, A. P. Ambler, I. Bellos, RAPT: A language for describing assemblies, Industrial Robot, 5(3):131--137, 1978.

[3] Smith, R. C. and Cheeseman, P., “On the Representation and Estimation of Spatial Uncertainty,”. The International Journal of Robotics Research, 5(4):56-68, 1986

[4] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner and S. Thrun, "Experiences with an interactive museum tour-guide robot", *Artificial Intelligence*, vol. 114, no. 1-2, pp. 3-55, 1999.

[5] Carvajal, D. (2017). Let a Robot Be Your Museum Tour Guide. Nytimes.com. Availableat: https://www.nytimes.com/2017/03/14/arts/design/museums-experiment-with-robots-as-guides.html [Accessed 5 Dec. 2017].

[6] The Verge. (2017). *LG’s new airport robots will guide you to your gate and clean up your trash*. [online] Available at: https://www.theverge.com/2017/7/21/16007680/lg-airport-robot-cleaning-guide-south-korea-incheon [Accessed 14 Dec. 2017].

[7] Richards, Hamilton. "Edsger Wybe Dijkstra". A.M. Turing Award. Association for Computing Machinery. Retrieved October 16, 2017. At the Mathematical Centre a major project was building the ARMAC computer. For its official inauguration in 1956, Dijkstra devised a program to solve a problem interesting to a nontechnical audience: Given a network of roads connecting cities, what is the shortest route between two designated cities?

[8] Shmyrev, N. (2017). *CMUSphinx Open Source Speech Recognition*. [online] CMUSphinx Open Source Speech Recognition. Available at: https://cmusphinx.github.io/ [Accessed 14 Dec. 2017].

[9] Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A*". *International Journal of Geographical Information Science.* 23 (4): 531–543. doi:10.1080/13658810801949850