Implementing A* and the Monte Carlo Localization algorithm with text-to-speech technology in the context of a food-delivery robot

Florian-Andrei Blanaru Andreea-Bianca Fraunhoffer Yuji Fukuta Rahul Gheewala Samuel Irsai

***Abstract*—Ordering food from a restaurant has become a very common practice, especially when considering the current circumstances, represented by the pandemic. Keeping clients in touch with the restaurant, while also minimizing physical contact between individuals and increasing the efficiency of the process could be achieved by automating some of the tasks involved. Our project proposes an implementation encompassing everything from placing your order to delivering it. To do this, we use text-to-speech technology, so a robot could speak to a customer and take their order, a particle filter for localization, so the robot can accurately self-localize on a given map, and A* for path-finding, so it can compute (and follow) the optimal path to a customer's delivery address.**

Index Terms*—automating, text-to-speech, particle filter, A

I. Introduction

The idea of automating tasks that are prevalent in the service industry is not new[5], and the demand for this has been steadily increasing since the pandemic began[3]. Limiting unnecessary physical contact between people by replacing the employee that would take a customer's order with a robot, as well as the delivery courier would prove to be advantageous not only because it promotes healthy social distancing practices, but also because the tasks would be executed more efficiently. Our project would still retain relevance after the pandemic, because the robot could work alongside human employees during peak hours at the restaurant, for example. The first task the robot should be capable of fulfilling is to take an order from a customer. For this, we are using text-to-speech technology; more specifically, the robot communicates with the customer through spoken words, and allows the customer the choice of responding either through keyboard or voice input. The information relayed by the client (details of their order such as sandwich fillings and type of drink, their delivery address) are stored by the robot, allowing for future use. Following that, the robot delivers the customer's order to the delivery address they specified earlier. This is achieved by using the Monte Carlo Localization (MCL) algorithm[6], as well as A*[1]. The robot figures out the optimal path from its starting position on the map (in the restaurant), and follows it. After dropping off the order, it either backtracks to the restaurant, or delivers the other orders in its queue by computing the optimal path to the other address, and so on. Our joint implementation of both a localization and a path-finding algorithm results in a robot that can efficiently and autonomously navigate through a predefined static environment.

II. Related work

In terms of expanding on pre-existing scientific work, we took great inspiration from research such as [2]. This paper explains how with the use of Genetic, Artificial Neural Networks and A* algorithms, it is achievable to move items from one location to another in a dynamic environment without collision. The paper provides the example of robots transporting carry tools in an industrial environment. We as a team thoroughly studied the report and found great value in its contents, in particular the A* implementation as we already had prior knowledge from our first-year AI module. Additionally, the research explains how A* heuristics excels in terms of time and speed when the size of this input is small. Since we plan in designing a map for ROS, this requirement would be satisfied. We imagined our delivery system to work for small local restaurants. Whilst we appreciated the work, we felt we could improve it by supplying the robot with a dynamic voice input from the user, which we considered to be our unique selling point. Although unsure if we can implement a dynamic environment in the given timeframe, by using A* we are able to keep this option open for the future. We also decided to adapt the method of localization by using MCL which we felt better suited our project style.

III. System/Framework description

A. Text-to-speech functionality

The text-to-speech functionality is implemented in our project through the SpeechRecognition, pyttsx3 and pyaudio libraries. The *get_audio* method processes the audio input from the user and translates it into text. If the input is not processable, the user will be prompted to speak again. The *take_option* method takes the resulting text and turns it into the list 'words', keeping each individual word as one element. These two methods lie at the base of all the subsequent methods, which follow a similar pattern: the user will be prompted to choose from a set of options and speak their answer; after the initial processing, the list containing the split input will be parsed looking for certain keywords which signals that a certain option has been picked. For example, when choosing whether the user is a customer or an employee, the algorithm is waiting for the presence of the keywords

'one', 'first', '1st' and 'customer'. When making a new order, the customer will be asked if they want to order a drink (and, if yes, to pick one from a selection of available drinks), and then questions about the type of bread and filling their sandwich should have. The end of this process is marked by the prompt the user will receive after giving all the details about their order, which will require them to type their address (A, B, C or D, when considering the environment described by our map). This user input will be transcribed in a text file which will be used as reference for the goal coordinates for the A* path-finding algorithm.

B. Localization using the Monte Carlo Localization algorithm

We have improved the performance of the particle filter we had to implement for Assignment 1. A brief description of how it works is as follows: the particle cloud is initialized using the *initialise_particle_cloud* method, which, given an initial pose guess, populates the particle cloud with particles, i.e. with hypotheses about the actual pose of the robot. *update_particle_cloud* updates the weights of the current particles by comparing the expected sensor data to the actual sensor data, creates new particles based on the previous iteration of particles with a high weight and discards the particles with a low weight. In the end, a new pose estimate is chosen through the method *estimate_pose*, which returns the particle with the highest weight. Our project makes us of a localization algorithm, because it allows the robot to accurately follow the path picked by A*. The initial pose will also serve as the starting location for A*.

C. Path-finding using A*

We decided that our contribution and one of the main features of our project is the implementation of an A* algorithm in ROS. To begin with, the helper method *fill_nodes* converts every pixel in the image of the world map to a node, which can be either traversable (drawn in white on the map), or non-traversable (drawn in black), while *fill_valid_neighbours* keeps track of every walkable neighbour of a given node. The method *a_star* takes two tuples as parameters: start(x, y) and end(x', y') which represent the pixel coordinates of the start and goal locations. The score of the start node is set to 0 (in the beginning, every node on the map is initialized with a very high score that will be reduced as the algorithm progresses), and the node is added to the list of nodes to visit. The main loop of the algorithm focuses on the size of the list of nodes to visit. While it is greater than zero, the first node will be popped (visited); if it is the end node, its coordinates will be added to the list of nodes in the path, and then the algorithm will backtrack through the visited nodes, adding the new pair of coordinates to the front of the list, which will be returned as the found path. If the current node is not the goal node, the algorithm will compute the new scores of every valid neighbour as the sum of the absolute values of the differences between the x and y coordinates of the end and current nodes. If the new score is smaller than the neighbour's previous score, its score is updated, the current node is added as its parent, and it is added to the list of nodes to traverse according to its score. The order in which the nodes are added is essential to the algorithm, because the nodes to be visited first should have the smallest score; otherwise, the algorithm would return sub-optimal paths. A* is the algorithm we use in our project to find the best path from the restaurant to a given delivery address. To better visualize the path, we have added a helper method *show_path_image*, which draws the map image pixels corresponding to the generated path in red.

D. Integrating in ROS

We tie all these components together (and actually run the robot) using two methods: *run_robot* and *localise_mcl*. *Run_robot* deals with the odometry readings, taking the A* generated path as a parameter; first, for every pixel in the path, it translates the image pixels to simulation coordinates using calibration formulas. It also computes the difference between the current coordinates of the robot and the coordinates of the goal, as well as the *angle_to_goal*. The last step before moving towards the goal is updating the robot's orientation given its angle, theta, by rotating at an acute angle, thus saving some time. *Localise_mcl* first waits for the robot to gather enough sensor information to accurately localise itself by rotating in place twice, and then feeds the estimated pose to the A* path-finding algorithm, as the start coordinates.

IV. Experimental Setup and Results

A. Testing the accuracy of our particle filter

During the experiments, three of the main features were tested. The first hypothesis is that the robot localises correctly in the provided map with a good estimated pose. The second hypothesis is that the user will be able to interact with the robot via text-to-speech recognition and via text input. The third hypothesis evaluated is that the robot will be able to deliver the order by using the shortest path calculated using A* algorithm. For evaluating the localisation, a PNG map was created using PowerPoint. In order to be used as a valid map in ROS, the .YAML and .PGM files were generated by modifying [4]. After 15 robot moves the results were recorded both visually(fig. 1-4) and in a table (table 1).

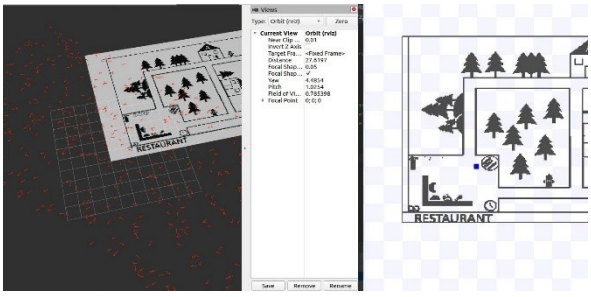

Fig. 1. The initialization of the MCL algorithm used in assignment 1

Our implementation of the MCL algorithm for the first assignment did not work properly for most of the cases (figures 1 and 2), returning faulty results (table 1) or taking a lot of

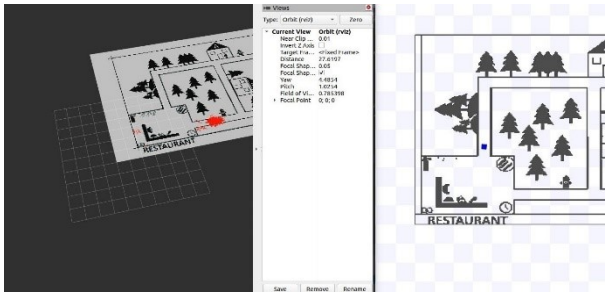

Fig. 2. The faulty result of the MCL algorithm used in assignment 1

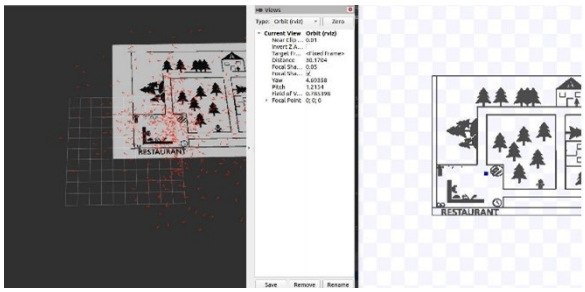

Fig. 3. The initialization of the improved MCL algorithm used in this project

time to localise the robot. Our improved MCL (figures 3 and 4) manages to localise the robot correctly (table 1) and in a much shorter amount of time, compared to the previous version. For the first MCL version, we observed an error between the x values of the position of 2.98, and an error of the orientation of 0.08, therefore a bad result overall. The current MCL has shown error of the x value of 0.32 and an orientation error of 0.25, which, as shown in the figures, outlines a more effective implementation of this algorithm. The differences between the two implementations lie in the initial population of particles, and in the way we normalized the weights of the particles. The first version used to allow particles to appear in a polygon which overlapped the map, whereas now we only let the particles spawn around the initial pose guess. The weight normalization (dividing every weight by the sum of every weight) was streamlined by storing the sum of all of the particle weights in a variable before iterating through the particles and normalizing them.

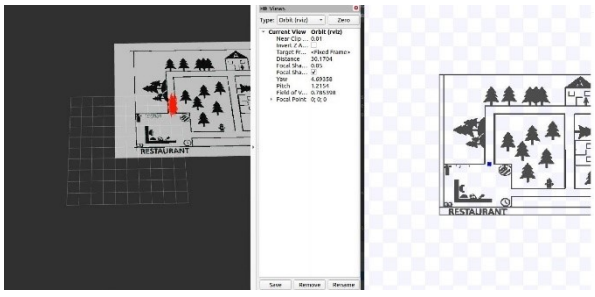

Fig. 4. The accurate localization of the improved algorithm used in this project

TABLE I. IMPROVEMENT OF THE MCL ALGORITHM FROM ASSIGNMENT 1 TO PRESENT

	Estimated pose		Actual pose	
	Position	***Orientation***	***Position***	***Orientation***
MCL Assignment 1	x: 4.779046058654785	x: 0.0	x: 7.751586033568981	x: 0.0
	y: 2.6731996536254883	y: 0.0	y: 0.9729899296250288	y: 0.0
	z: 0.0	z: -0.99986470762149	z: 0.0	z: -0.991331188242391
MCL now	x: 4.684895038604736	x: 0.0	x: 4.365258729976629	x: 0.0
	y: 4.716181755065918	y:0.0	y: 4.272829072256103	y: 0.0
	z: 0.0	z:-0.9988394391213407	z: 0.0	z:-0.746774807990727 7

B. Testing the accuracy of the text-to-speech assistant

For evaluating the interaction between the user and the robot via text-to-speech, the script was tested against different audio inputs from various people and microphones. The phrases to be checked were generated randomly using [9]. The accuracy of the voice interpretation was compared based on the output from the robot and it is stated in the following table (fig. 2.0). The time comparison between both methods was executed using a stopwatch and different speed inputs. The interaction via only text input worked every time the user added a valid input. Although the accuracy of the text input is much higher than the text-to-speech, the time differences are observed in the table below (fig. 2.0).

Judging from our experimental results, we were surprised at the accuracy scores and felt more confident in our existing approach. We calculated using our results, the total average accuracy for speech was 86.6% but for typing it was 99.6% but the time costs were 4.02s and 12.11s respectively . We found that despite providing a complex array of sentences, the google speech recognition performed better than expected and we felt it had potential for further improvements for the purpose of our project. Given that the accuracy is relatively high on most examples, we decided to go ahead with an approach in which certain keywords are extracted from the user to distinguish between different options. It is risky in the sense that if this keyword is not picked up, the decision making process would fail. But given the high average accuracy, we felt the likeliness of this keyword being missed would not be significant. Furthermore, we surround each case with a try/except to ensure if a keyword is not recognized, the user can continually attempt to provide input until the required criterion is met. We as a group felt the addition of Speech-

TABLE I
THE ACCURACY AND THE SPEED RESULTS FOLLOWING THE TESTS

User input	Output Recognition	Accuracy Text-to-Speech	Accuracy text input	Time to use Text-To-Speech	Time to write text input
There can never be too many cherries on an ice cream sundae.	In hopes of finding out the truth, he entered the one-room library.	100%	100%	3.52s	9.92s
Joe discovered that traffic cones make excellent megaphones.	Joe discovered that traffic cones big excellent megaphones	87%	99%	4.20s	17.52s
Trash covered the landscape like sprinkles do a birthday cake.	Trash covered the landscape like sprinkles do a birthday cake	100%	100%	4.23s	11.09s
With a single flip of the coin, his life changed forever.	With a single flip the coin his life changed forever	90%	100%	3.40s	12.27s
The newly planted trees were held up by wooden frames in hopes they could survive the next storm.	The newly planted trees were held up by wooden frames in home so good survived the next door	77%	99%	5.86s	19.60s
He spiked his hair green to support his iguana.	Despite his hair green to the port is iguana	55%	100%	3.14s	8.00s
The spa attendant applied the deep cleaning mask to the gentleman's back.	the spot attendant applied the deep cleaning mask to the gentleman's back	92%	100%	4.27s	11.54s
The hawk didn't understand why the ground squirrels didn't want to be his friend.	the Hulk didn't understand why the ground squirrel didn't want to be his friend	85%	98%	4.86s	13.93s
He always wore his sunglasses at night.	He always wore his sunglasses a night	85%	100%	2.51s	6.45s
In hopes of finding out the truth, he entered the one-room library.	In hopes of finding out the truth he entered the one-room library	95%	100%	4.21s	10.78s

Fig. 5. The accurate localization of the improved algorithm used in this project

to-text was a critical feature to aid the user experience of our project, and would help to enhance our project which could have commercial applications in the future. Other reasons included ease of use, as shown in our experimental results, we found in all cases that using speech was significantly faster than using text input. Although for speed there was a cost in accuracy, we understood during the course of our project that this was not essential. As long as the key points were extracted from the information, the algorithm we implemented did progress in the decision guiding progress.

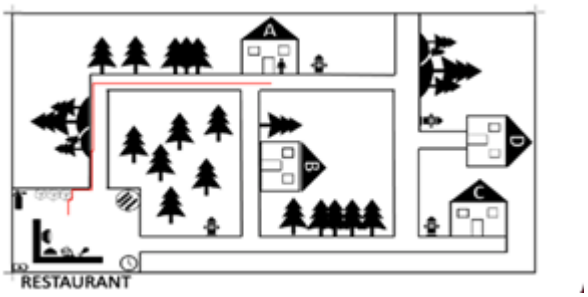

Fig. 6. A* path generation for house A

C. Testing the functionality of the A* path-finding algorithm

For evaluating the third hypothesis, before testing it in ROS, the implementation of the A* algorithm was tested with various inputs for the starting waypoint and the ending waypoint in the map. For a better analysis of the results, an image with changed pixels(red coloured) based on the generated path was generated and represented in the following screenshots (fig. 6-9).

As observed in the images, the algorithm works very well for the selected waypoints representing the starting point of the robot and also the ending points which are the houses. It successfully manages to come up with the best path for the robot to follow.

When integrating A* in the MCL implementation on ROS, we observed that the path-finding algorithm picks paths that follow the outline of the non-walkable map pixels. This caused the robot to hit an obstacle while trying to take sharp turns and stop moving in the simulated environment, even though the path was optimal (fig. 9)

Our fix for this issue was to check for non-walkable pixels in the current pixel's immediate neighbours; if there is a single

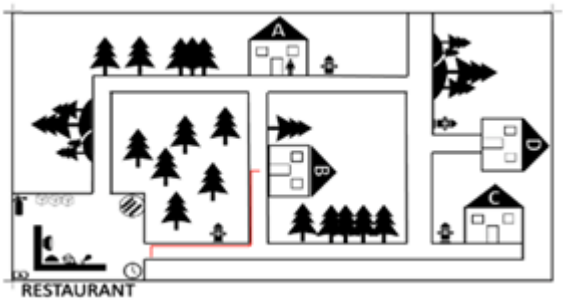

Fig. 7. A* path generation for house B

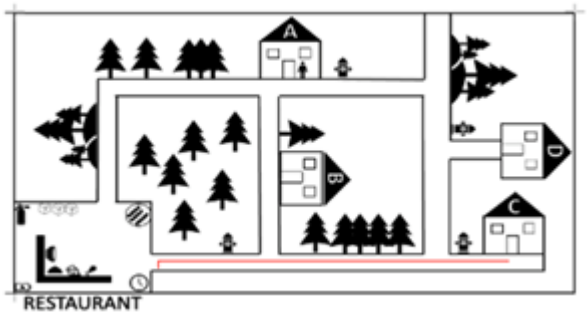

Fig. 8. A* path generation for house C

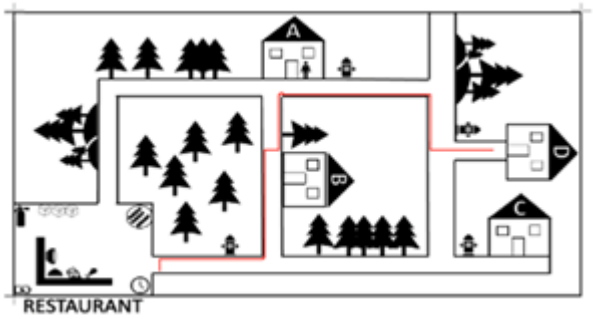

Fig. 9. A* path generation for house D

Fig. 10. The robot hitting an obstacle while following a path

obstacle anywhere in the desired interval, then the current pixel itself is redefined as a non-walkable pixel, and A* is required to recompute the trajectory of the path. This fix is in the *map_a_star* method.

One other issue we encountered during development was that our initial calculations for the translation from pixels to simulated coordinates proved to return misleading results which created a scenario where the robot would not successfully follow the path, sometimes leading it into a wall. We resolved this problem by adapting the formula to the ROS map simulated coordinates, which in the end, gave the robot a higher efficiency in running through the actual path given by A*.

V. Conclusions and Future Work

We gave careful thought to what other additional features our project could have benefitted from, or ways to polish the features that were already implemented. An idea which we consider is to implement everything in a dynamic environment, as we saw in [2], rather than a static one. That is, ensure that the robot is able to detect changes in the environment, and reacts accordingly. Another idea to consider would be air delivery. Delivery drones have already been implemented[7], and we would be curious to try and tailor A* for airborne conditions. In conclusion, our project has proposed an implementation of speech-to-text recognition technology, a particle filter based on the MCL algorithm and A* heuristics. The robot is able to take orders from customers and deliver them from a restaurant, this would prove to be a helpful alternative to human employees, especially in the context of the pandemic.

References

[1] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107. doi:10.1109/TSSC.1968.300136.

[2] R. Kala, A. Shukla, R. Tiwari, S. Roongta, R. R. Janghel (2009) Mobile Robot Navigation Control in Moving Obstacle Environment using Genetic Algorithm, Artificial Neural Networks and A* Algorithm, Proceedings of the IEEE World Congress on Computer Science and Information Engineering, Los Angeles/Anaheim, USA, pp 705-713

[3] Hobbs, J. E. (2021). Food supply chain resilience and the COVID-19 pandemic: What have we learned?. Can J Agr Econ. 2021; 69: 189–196. https://doi.org/10.1111/cjag.12279

[4] Sperbeck, C. (2016). MakeROSMap source code (Version 1.0) [Source Code].https://drive.google.com/file/d/0B2AcDRX3bKLVdjhPU1B2UUNRaDA/view?resourcekey=0-fKWCpCO1UiiQgNNGGzMlPA

[5] Ivanov, Stanislav Hristov and Ivanov, Stanislav Hristov and Webster, Craig and Berezina, Katerina, Adoption of Robots and Service Automation by Tourism and Hospitality Companies (May 6, 2017). Revista Turismo & Desenvolvimento, 27/28, 1501-1517., Available at SSRN: https://ssrn.com/abstract=2964308

[6] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots." Proc. of the Sixteenth National Conference on Artificial Intelligence John Wiley & Sons Ltd, 1999.

[7] BBC News. 2021. Amazon makes first drone delivery. [online] Available at: ¡https://www.bbc.co.uk/news/technology-38320067¿

[8] The team's repository: https://gitlab.com/group-07/intelligent-robotics/

[9] Random word generator: https://randomwordgenerator.com/sentence.php