Mobile Robot Modelling and Navigation in an Unstructured Environment

180195046

University of Sheffield, Sheffield, S10 2TN

ARTICLE INFO

Article history:
Received 21 May 2021

Keywords:
Robtics, Path planning, Kinematic models

ABSTRACT

This paper investigates pathfinding algorithms and simulates kinematic models in mobile robotics. It compares three different pathfinding algorithms around five different scenarios comparing the success of establishing a path, the path distance and the time taken to calculate the path. It was found that the probabilistic roadmap was the most reliable method and produced the shortest path. The Rapidly Exploring Random Tree was the quickest method but lacked consistency or convergence. The user created genetic algorithm had some technical limitations resulting in a slow and ineffective performance. Once a path had been created this was tested using a simulated kinematic model and this was shown to be representative of a real-world robotic test.

1. Introduction

Robotics and automation are quickly becoming a major part of many industries especially as the world enters the fourth industrial revolution. Mobile robots are a large subsection of robotics and are defined as the study of any robotic system that can move within its own surroundings. Most mobile robots are autonomous meaning that they can navigate their environment without an input from humans. These types of robots are essential for when there are places that humans can not get to, such as dangerous sites like mars or at nuclear accidents. Increasingly mobile robots are being used in both industrial and medical settings where they can move materials and equipment quicker and more efficiently than humans.

This however brings up a key social and ethical issues with robots and automation. One such issue is the replacement of human workers. Whilst it could be argued that robotics and automation allow humans to avoid repetitive and tedious tasks and take on more creative roles. There is a large majority of people in society see them as competition to their jobs. Backlash to robotics is common and this is one of the main costs associated with robotics as it is expensive to retrain human operators and current systems to work with robot systems. This factor will hopefully be solved with time and when more people are educated with the advantages and limitations of robotic systems.

The aim of this report is to research and compare path finding algorithms that some mobile robots use to navigate, and the kinematic models required to simulate a mobile robot in a virtual environment. The benefit of simulating mobile robots is that researchers to focus on developing efficient algorithms and models without having to worry about hardware constraints or real-world errors. Once methods are perfected in a simulated environment they can be tested in the real world and any differences compared and solved.

2. Background theory

2.1. Path planning algorithms

Path planning is the process of finding a sequence of valid configurations that move an object from a specified start location to the end location. This is done by providing the system with a domain with a predefined set of constraints and a goal (such as walls and an end point). The algorithms covered in this paper all divide the domain up into discrete locations then solve these through different computational methods.

2.1.1. A star

The A star algorithm is a very widely used algorithm and it is used both in direct pathfinding and in both the PRM and RRT methods. The algorithm works by producing a tree of paths starting from the start point and extending the paths one node at a time until a criterion is reached (the end point). Every iteration it determines which path to extend by looking at the cost of a path and estimating the cost required to extend the path all the way to the goal. The path with the lowest cost (shortest distance) is then extended until the endpoint is reached. (MathWorks, 2021)

2.1.2. Probabilistic roadmap (PRM)

In this approach a path is found by taking random samples of the workspace and checking if they are valid positions for the robots to travel (ie they're not a wall). Once multiple positions have been found they are connected using lines within defined variables (either line length or a selected number of closest dots). Once all the lines have been drawn and start and goal positions are defined for the map a query is run. If there is a path all the way from the start to end dot it is found using the A star algorithm, if not the path returns a nul value. (Korkmaz & Durdu, 2018)

2.1.3. Rapidly-Exploring Random Tree (RRT)

A rapidly- exploring random tree is a probabilistic based method used to map out areas using a iteratively expanding random tree. The RRT grows a rooted tree out from a selected start point. Each time a new point is chosen using specific inputs (such as length from previous point) and connection attempt to the previous point is made. The connection is checked against a validation criterion for example is there a wall blocking the connection and only if the connection is valid it is added to the tree. The algorithm uniformly searches the whole allowable space and decides the probability of expansion depending on the size of the Voronoi region. As this ensures that the areas at the edge of the tree have the largest Voronoi region and are therefore explored first. Once the tree reaches the specified area of the goal point the tree creation stops and the quickest route from the start to finish is found. Another variant of this method is the Bidirectional Rapidly Exploring Random Tree (BiRRT) which branches from both the start and endpoints and creates a path once both trees meet. (Korkmaz & Durdu, 2018)

2.1.4. Genetic Algorithm (GA)

A genetic algorithm uses biological principles of evolution and survival of the fittest to iteratively produce better results over many generations. To start with a genetic representation of the problem domain is created for each individual in a population. The population then is tested against the problem with a fitness function calculated for each individual determining their success rate. Those individuals with a higher success rate are more likely to be chosen to mutate and these offspring are then slightly modified (either mutated or combined with each other) to create the next generation. Over time and multiple iterations, the overall population should have a higher average fitness function and therefore be better at the required task. (Korkmaz & Durdu, 2018)

2.2. Sensors

Range sensors area used commonly across robotics. They act as an input device allowing a robotic system to 'see' its environment so it can compute logical actions and execute them. The sensor used in this report is the lidar sensor and this sensor is used to determine the distance to an object such as a wall or obstacle in the robot's path. It works by firing a laser and measuring the time it takes for the laser to reflect of the object and return to the receiver.

2.3. Kinematic models

Kinematic models are used to simulate the dynamics of real-world vehicles in a virtual space. They take inputs such as the pose (the cartesian location of the robot and its current direction) the directional and angular velocity and the location of the next waypoint. From these they calculate outputs such as the acceleration and wheel angle needed to reach the next way point. The three models covered in this paper are the differential drive model the unicycle model and the bicycle model.

The differential drive model approximates a vehicle with a single fixed axle with wheels of a specified radius separated by a specific track width. Both wheels on this model can be driven independently and can change to any angle in the specified range. The vehicle speed is defined from the center of the axle. The unicycle model is similar to the model above but is defined as a single wheel with a specified wheel radius that can move to any angle in the specified range. The bicycle model resembles a traditional bicycle with two axles separated by the specified distance wheelbase. In this model only the front wheel can be positioned independently by an angle of psi, with the back wheel turning by an angle specified by the vehicle heading value, theta.

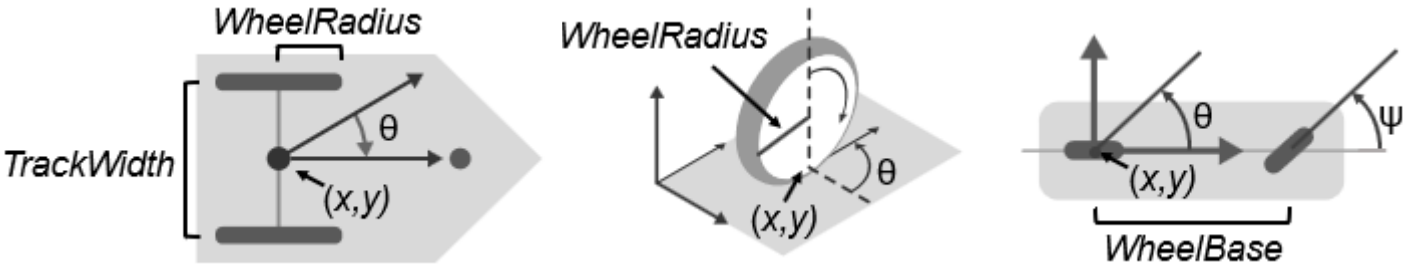

Fig. 1 - Differential drive (left), unicycle model (middle), Bicycle model (right). (MathWorks, 2021)

3. Implementation

The algorithms and code were first researched and developed in MatLab then transferred to the MatLab App Designer so they could be displayed on the GUI. The overall structure of the of the code is summarized in figure 2 with each section of the code covered throughout this chapter.

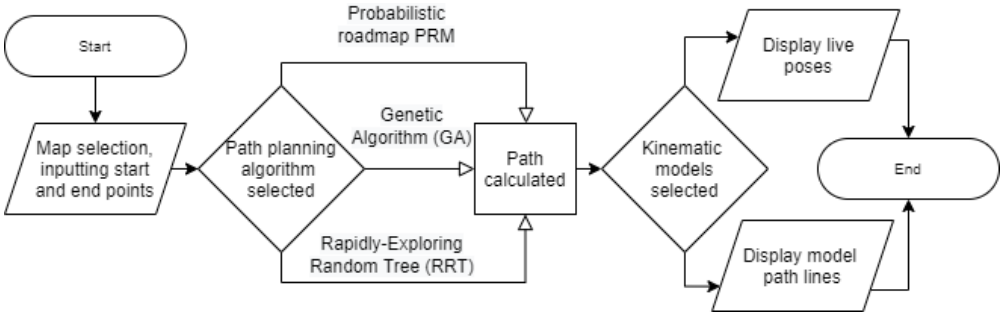

Fig. 2 – Flow chart showing the key structure and decisions of the code.

3.1. Map selection and inputting start and end points

The first stage is map selection this is shown in lines 970-1030. In this stage the user selects a map from the drop-down menu (shown in figure 3). The map, an inflated version of the map and a generated binary occupancy map are then all saved under global variables. (app.map, app.non_inflate_map, app.binaryOccupancyMap). The lines from 1014 then display the chosen map in each of the figures on the GUI. Lines 1026 create a new blank map using the limits of the chosen map to be used in the sensors figure.

Once the map is chosen the start and end points can be inputted. This is shown on lines 954 and involves the user inputting start and end coordinates into the numerical input boxes shown in figure 3. These points are then stored in global variables (app.startLocation and app.endLocation) and plotted as red and green dots on the main map.

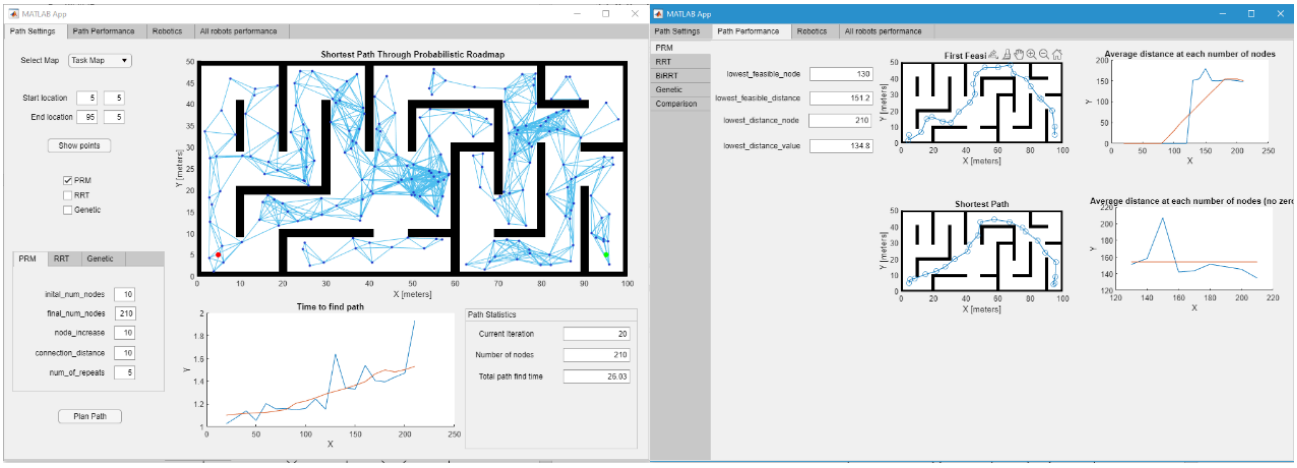

Fig. 3 – Main page of GUI with settings (left), path analysis page (right).

3.2. Probabilistic roadmap PRM

The first pathfinding technique is the probabilistic roadmap (or PRM), this is shown in lines 203 – 446. This method involves the prebuilt prm1 function to plot randomized nodes on a graph and find connections between nodes within a specified distance (shown on line 265). Using the prebuilt findpath function (line 274) to attempt to plot a path between the specified start and end points using the made connections and the A star algorithm.

A custom distance function was created to calculate the distance of the path. This was done by first calculation the number of (x,y) values in path then looping through each set of coordinates and calculating the difference between each component and the last. Using this difference and Pythagoras' theorem the hypotenuse was calculated and added to a rolling total called distance which was outputted once all the coordinates had been used.

The process for the implementation of this function within MatLab and is provided in the flowchart shown in figure 4. The input variables that are shown in the bottom left of figure 3 control the settings for the program.

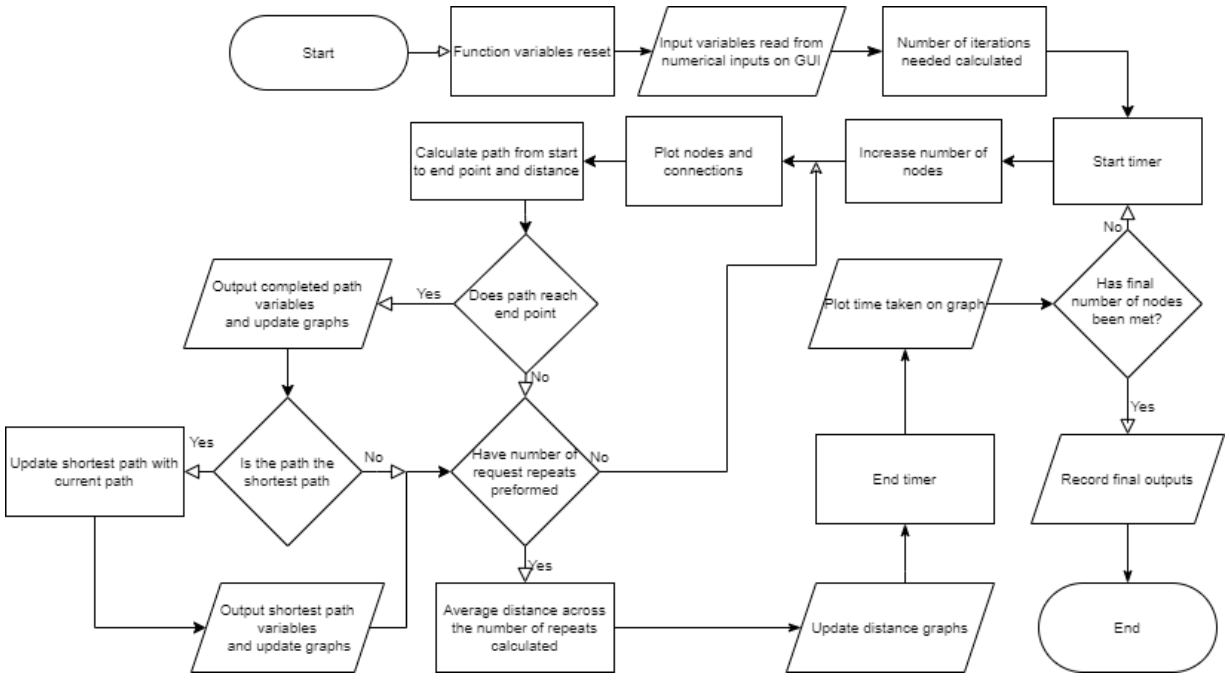

Fig. 4 – Flowchart showing the key processes for the PRM and RRT algorithms.

3.3. Rapidly-Exploring Random Tree (RRT)

The rapidly-exploring random tree is shown in lines (450-707). The process used for this algorithm is similar to the probabilistic road map and follows the same flow chart as the PRM (figure 4). The aim of this function is to plot a multibranch tree through the map with the higher number of branch iterations leading to more reliable path generation and shorter distances.

This algorithm uses prebuilt functions to calculate and plot the RRT map. These include generating a state space and a state validator map (477-488) that acts as a occupancy map and a validator to ensure that the plotted branches are valid (i.e. not going through walls or out of the maze). The function then generates a planner (490) to perform the calculation of the route with the variables inputted in the numerical input on the GUI (lines 491-493). The code then follows the same structure as the PRM until line 535 where the rng('shuffle') function is used to generate randomized paths each time. The function plan (line 537) plots the points, calculates the connections then uses the A star algorithm to find the shortest route through the maze. The rest of the algorithm is outputting variables and plotting the first feasible path the shortest path, the average distance each iteration and the time taken to perform.

3.4. Genetic Algorithm (GA)

The genetic algorithm was a custom written code to looking at whether a self-improving algorithm could outperform the two previously mentioned methods in terms of path distance. The fitness function used in this instance was the distance to the goal and once the goal had been reached the distance of the path length. This ensured that the dots that were closest to the goal would be selected to mutate for the next generation. The selected dots to continue had multiple mutated clones made but the top dot would stay to ensure that the overall population couldn't mutate further away from the goal.

The main procedures in the algorithm are show in the flow chart in figure (5) and match with the code used on lines (711-945). Key functions used that were already prebuilt in MatLab include both the randi function (line 746), the randsrc function (line 882), both used to generate random numbers between a specified range. The mink function (line 840) was used to choose the specified 'k' number of lowest values in a matrix.

3.5. Robot selection and live display

Once a path has been generated using the path finding algorithms it is saved as a global variable. This path is then used as points to be followed by the robots. The first part of the robotics tab is the live displays. Lines 1126 to 1149 show the code for when the select robot dropdown menu is selected. For each option a prebuilt MatLab function including the robot's kinematics (simulating the dynamics of the chosen robot) and controllers (a geometric controller used for following paths) are selected about input variables preset. These are then saved as global variables to be sent to the next part.

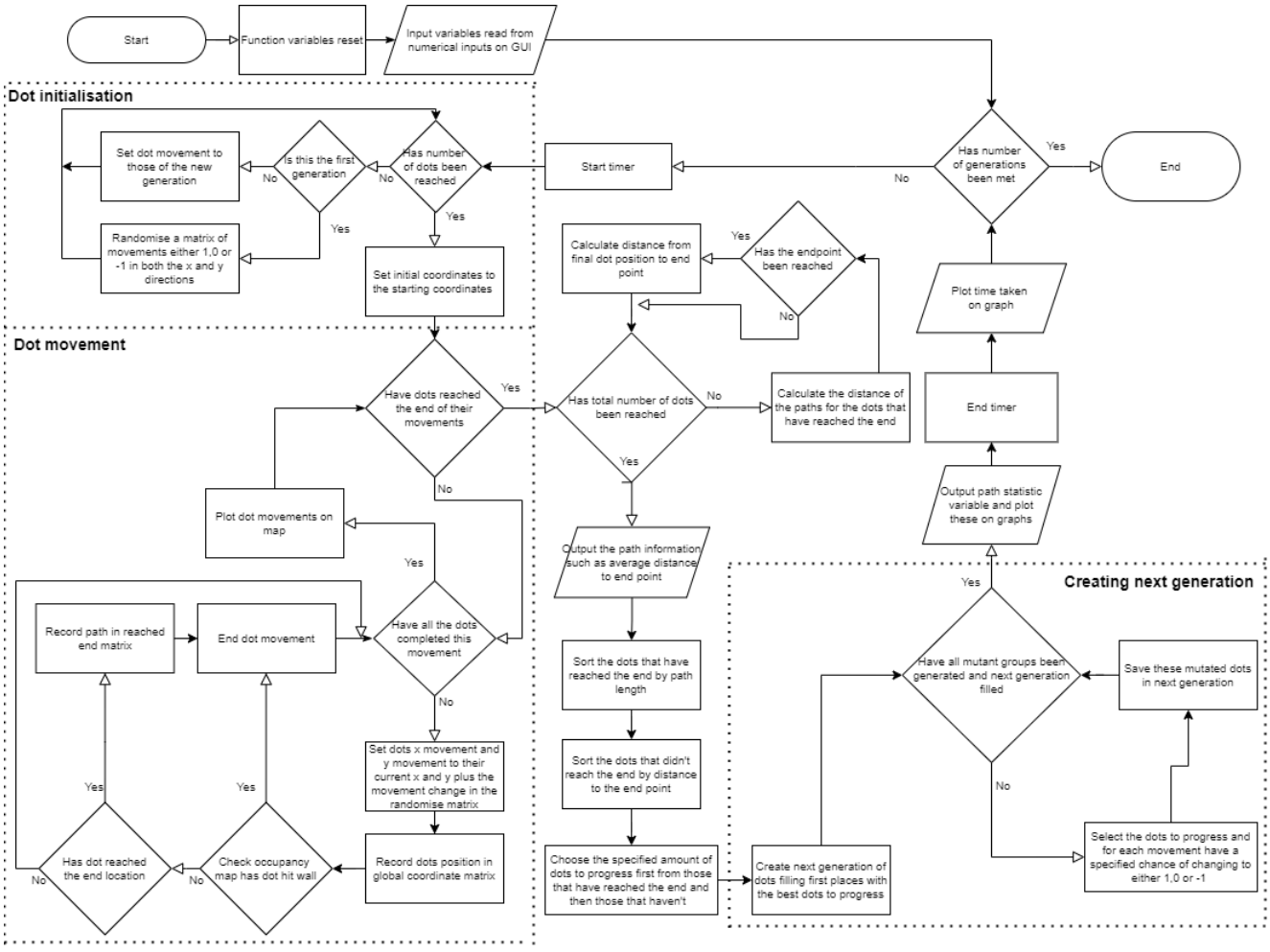

Fig. 5 – Flowchart showing the key processes for the genetic Algorithm.

After the robot has been selected line 1036-1120 show the code require to display live positions of the robot and sensor data. The code starts by setting up sensors using the inbuilt function rangesensor with a given range. The maps are then all cleared (1040) and a time matrix is set up to control the speed at which the robot moves and graphs are updated (1045). Then an initial post of the robot is created using the first coordinates from the selected path (1049) and the path and end point is saved using the app.controller.waypoints function (1052). Then the map is shown on the figures with the path displaced on the first map and the last sensor map left blank (1058).

Then a for loop is initialized that loops every time period (as set above). The position of the robot is then set (1073) and compared to see if the robot has reached a specified distance away from the end goal. If it has the loop ends at the robot has reached its goal. If the robot hasn't reached the goal the sensor map is updated (line 1085) using the sensor function again with the lidarScan function to calculate sensor reading of the walls. These scans are then validated on line 1087 and then used to update the map (line 1089).

The pure pursuit controller is then used to calculate wheel speeds (1092) and the velocity of the robot is calculated using discrete integration (1095). This is then used to calculate the new locations of the robots and this is then plotted on the live map and sensor map using the inbuilt functions plotTrvec and plotRot and plot transforms. The images of the previous plots are also deleted in this step.

3.6. Model path lines

This tab allows all three robot paths to be plotted on the same figure. The code starts at line (1159) by setting up a time array, initializing poses and specifying goal points and radius. As in the live display controllers are specified but this time a separate controller and kinematic model is specified for each robot at once to allow for them all to be solved (line 1171). The map is then initialized with the selected map and the selected path drawn on. A checkbox system is set up at the bottom of the GUI that allows the user to choose which robots' paths they want calculating and plotting. For each selected robot the program then uses a ordinary differential equation solver (function ode45 on line 1090) to solve the dynamic problems of the robots movement and outputs the poses of the robots. These poses are in a 3D rotation form and are converted to plottable cartesian coordinates by the axang2quat function on line (1192). Theses poses are then plotted on a graph with the indicated color (shown in figure 10).

4. Results and Discussions

All three of the selected algorithms (RPM, RRT and genetic) were experimented with for each of the five selected scenarios. The main test was to compare the length of the shortest path each algorithm could produce. This was done by incrementally increasing a key factor in the cases of RPM and RRT (the number of nodes and the number iterations respectively). For the genetic algorithm, the number of generations was experimented with until a suitable result was achieved within the time constraints. The time taken to reach this final path length and the length and time to reach the first feasible path was also recorded.

The variables for the RPM were an increase of 10 nodes per iteration with 10 starting nodes and finishing at 1010 nodes. A max connection distance of 15 was used and number of repeats was set to 1 (no repeats). For the RRT an increase of 10 branch iterations was chosen with the algorithm starting at 10 starting iterations and finishing at 2000 nodes. A connection distance of 15 was used with a max number of tree nodes set to 1010 and repeats set to 1 (no repeats). The genetic algorithm was set to 2000 independent dots with a max movement value of 200. There were to be 30 generations with a 20% chance of mutation and 200 dots to progress each generation.

4.1. Simple map

The simple map was provided as a built-in map on the MatLab tutorial page as is used as a simple test map for the path planning algorithms. A simple test with only one corner to travel around had starting start coordinates of [3,3] and ending at [23,23] (shown in figure). A harder test with multiple corners and sharp turns had starting coordinates of [3,3] and finishing coordinates of [23,3].

Table 1 – Results from analysis of path planning algorithms on the simple map.

	Simple map easy: start [3,3] end [23,23]				Simple map hard: start [3,3] end [23,3]			
Path planning algorithm	**First feasible path distance (m)**	**Time to reach first feasible path (s)**	**Shortest path distance (m)**	**Time to reach shortest path (s)**	**First feasible path distance (m)**	**Time to reach first feasible path (s)**	**Final path distance (m)**	**Time for distance to converge (s)**
PRM	34.8	1.1	29.2	21.9	39.3	0.7	33.5	31.4
RRT	34.8	0.6	29.8	4.5	54.3	1.1	38.0	11.6
GA	85	525	49	1507	N/A	N/A	N/A	N/A

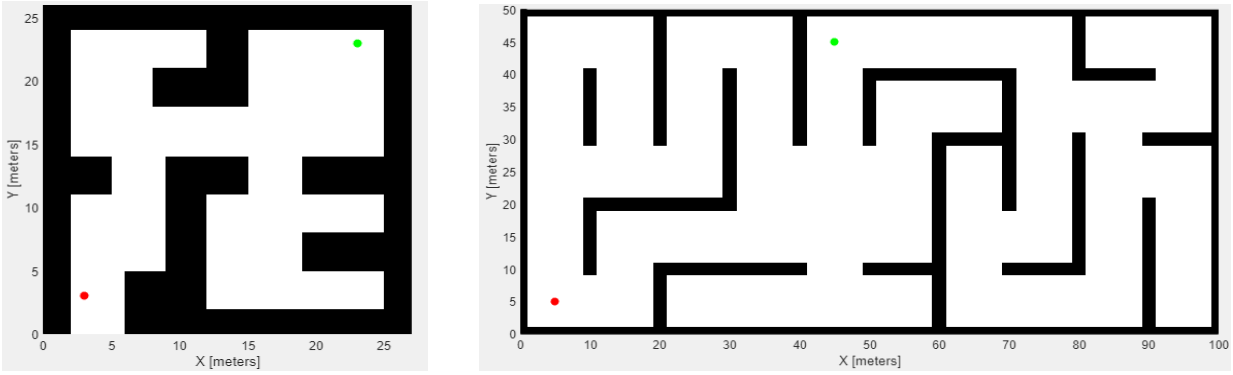

Fig. 6 – The simple map (left) and task map (right) with the easy and halfway start points (red) and end points (green).

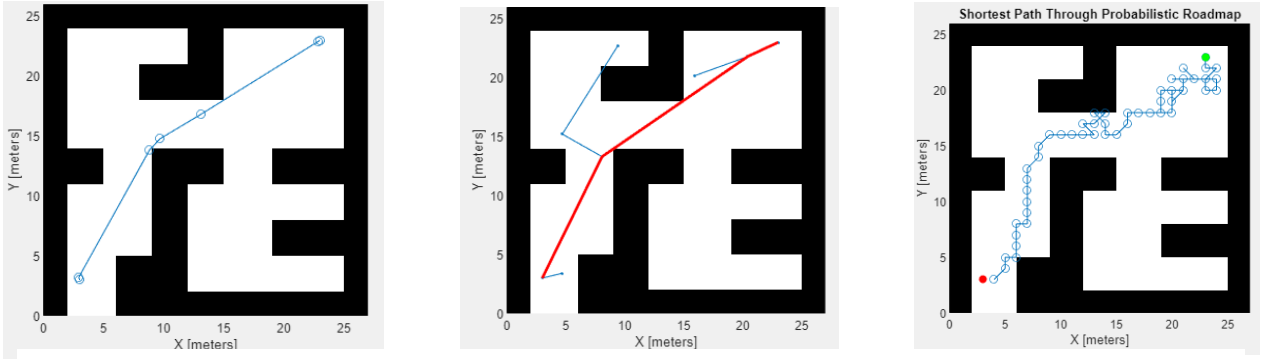

Fig. 7 – The PRM (left), RRT (middle) and GA (right) final paths for the simple map easy test.

4.2. Task map

The task map was created using excel and was more complicated than the simple map. The start locations for both the scenarios based on this map were [5,5] and the halfway coordinates were [45,45] (shown in figure 6) and end coordinates were [95,5].

Table 2 – Results from analysis of path planning algorithms on the task map.

	Task map halfway				Task map end			
Path planning algorithm	**First feasible path distance (m)**	**Time to reach first feasible path (s)**	**Shortest path distance (m)**	**Time to reach shortest path (s)**	**First feasible path distance (m)**	**Time to reach first feasible path (s)**	**Final path distance (m)**	**Time for distance to converge (s)**
PRM	76.6	1.4	61.1	42.0	139.3	5.2	127.4	84.0
RRT	79.0	0.8	60.0	17.2	162.7	5.1	139.2	17.1
GA	138	1689	97	4538	N/A	N/A	N/A	N/A

4.3. Complex map

The complex map was generated using an online map generator and was used as an advanced test as it was large with lots of different routes. The start location on this map was [5,5] and ended at [315,315]. The test variables for this map were altered to an increase of 100 nodes/iterations and a final node/iteration count of 4010 as the map was too large for the previous conditions to compute a valid path.

Table 2 – Results from analysis of path planning algorithms on the complex map.

	Complex map			
Path planning algorithm	**First feasible path distance (m)**	**Time to reach first feasible path (s)**	**Shortest path distance (m)**	**Time to reach shortest path (s)**
PRM	2610	29.6	1073	137.3
RRT	N/A	N/A	N/A	N/A
GA	N/A	N/A	N/A	N/A

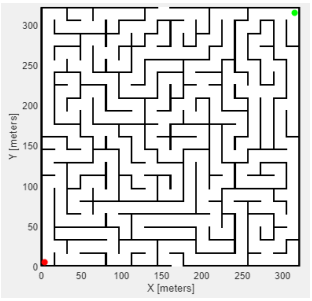

Fig. 8 – The complex map with start and end points

4.4. Pathfinding algorithm analysis

The results from all the simple and task map tests show that the PRM is slightly slower than the RRT at finding its first feasible path and converging on a final shortest path. Both techniques achieve either similar path lengths for the first feasible path or the PRM having a lower path length. The PRM finds an overall shorter path length. This shows that the RRT technique is quicker but less consistent, and it is unlikely to optimize the path length even when additional branches are added. This could be most useful in real world scenarios where the overall length of the path is not as important but the time to find the path is. However, PRM is more consistent at finding a path and would be a better choice in an application where ensuring a path was found is essential or where a shortest path is needed.

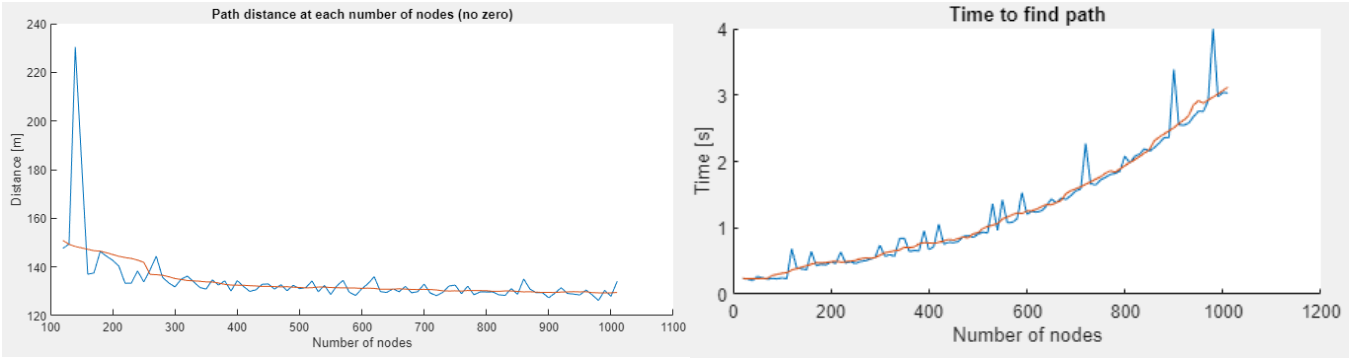

Fig. 9 – Graph showing the path distance converging(left) and the time increasing (right) for an increasing number of nodes using the PRM algorithm.

Both paths increase in time taken to compute exponentially as more nodes are added making these algorithms very computationally expensive when looking at larger mazes. This is true in the case of the large map where the PRM took a long time to solve and the RRT failed to find a path. The narrow paths with multiple branching dead ends made the RRT very inefficient and even with additional tests with up to 10,000 nodes failed to solve. They also both reach a point of saturation for any given map where no matter how many more nodes are added the path would only get marginally shorter. This is shown be low in the graphs showing the path distance against number of iterations (removing any zero results for clarity). A program could be set up to work out once the path has reached this convergence point and at this point the program could stop.

The custom created genetic algorithm was outperformed by the other two inbuilt functions. On the tests that the GA did reach the goal it was much slower than the other algorithms and produced much longer paths however this, but this was primarily due to the code not being optimized. With further code optimization such as random selection of 0.2 of movements chosen to be mutated instead of each movement having a random chance to mutate would have saved time as the code would not need to calculate random numbers for each movement. An issue found with this version and with all GA's was its ability to converge towards a local minimum instead of the overall problem's minimum. This took the form of all the population getting stuck with the goal on the other side of a wall but not able to go round as to go round the wall the individual would have to be further away from the goal on its first attempt and therefore would not get selected to progress to the next generation – this occurred on both the simple map hard test and the task map hard test resulting in this paths failure to reach the endpoint. A trial solution of implementing a fitness function that randomly choose individuals from the entire population, with a higher chance of selection to those with a higher fitness function, instead of just mutating the best individuals. This method was experimented with and fell into the same local minima issue. Another solution could be to combine the GA with another algorithm for example, giving the coordinates produced by the RPM algorithm and then the GA could optimize this path.

4.5. Kinematic model's results and analysis

All the kinematic models performed very similar to each other as the models were not too complex. They all had the same speeds so in a straight line were identical. When a turn had to be made all the models had the same angular velocity, so the only difference was the positioning of the wheels. This can be seen in figure (10) where both the differential drive model (green) and the unicycle (red) follow such similar paths that their path lines overlap each other meaning one of the lines cannot be seen. This is because both the unicycle can move its wheel at and angle its given range and as the differential drive's wheels are independent from each other they can too. This results in the maximum control and the smallest turning circle possible within the given angle constraints. However, the bicycles blue line can be seen on the figure as it has a larger turning circle since its wheels are not independent, so it takes longer the back wheel to change to the same angular position as the front wheel. This slightly larger turning circle means the bicycle takes longer to complete the path and is more likely to collide with a wall as it follows the route. Overall, all the different models successfully reached the end point.

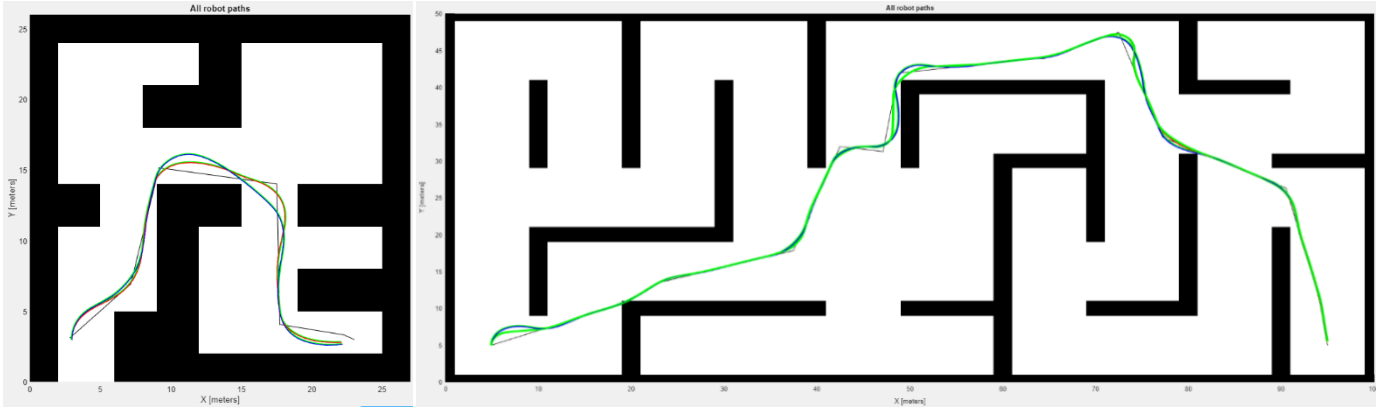

Fig. 10 – Path lines of the differential drive (green) unicycle(red) and bicycle (blue) on the simple map (left) and task map (right).

5. Conclusion

The experiment showed the use of path planning algorithms in providing a reliable method of navigation for a mobile robot. Of the algorithms looked at both the RPM and RRT provided quick and reliable paths, something that is of paramount concern for a robot in a real-world environment. There are various enhancement methods available to improve on these algorithms such as the biRRT algorithm. The GA was slow and inefficient but it showed potential as a way of optimizing paths especially if implemented in tandem with other algorithms. Another improvement to the real-world implementation of these algorithms would be to factor in different speed controlling variables such as a rough surface slowing a robot down and making sure the algorithm chooses the quickest route with these variables in mind.

The Kinetic models tested were basic but proved that the paths that the algorithms found could be followed successfully and that more work needed to be done when planning paths to ensure that the models would not crash due to wide turning circles and sharp corners. This could be implemented into the path planning process to achieve better results and would be essential when taking the algorithms into real world scenarios.

REFERENCES

Korkmaz, M., & Durdu, A. (2018). Comparison of optimal path planning algorithms. *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)* (p. 255). Aksaray: Aksaray University.

MathWorks. (2021, May 20). *Mathworks*. Retrieved from https://uk.mathworks.com/