Indian Journal of Science and Technology, Vol 9(45), DOI: 10.17485/ijst/2016/v9i45/102431, December 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Mobile Robot Navigation using Amended Ant Colony Optimization Algorithm

Velappa Ganapathy[1*] Priyanka Sudhakara[1] Titus Tang Jia Jie[2] and S. Parasuraman[3]

[1]School of Computing, SRM University, Kancheepuram - 603203, Tamil Nadu, India;
ganapathy.v@ktr.srmuniv.ac.in, priyanka.k@ktr.srmuniv.ac.in
[2]Computer Software, Monash University, Melbourne, Australia;
Titus.Tang@monash.edu
[3]Monash University, Lagoon Selatan, Bandar Sunway, Subang Jaya - 47500, Selangor, Malaysia;
s.parasuraman@eng.monash.edu.my

Abstract

Objectives: This paper presents an amended Ant Colony Optimization (ACO) algorithm for a mobile robot navigation to find the most optimal path. **Methods:** A modified design and development of an improved Ant Colony Optimization algorithm based upon a prior research work done is proposed in this paper. The algorithm put forth is enhanced by simplifying the equations already proposed and enlarging the area of the simulation framework, extending the task capabilities of the robot, as well as testing the algorithm in real time on an autonomous mobile robot. **Findings:** The proposed algorithm has to calculate optimal trajectory for the mobile robot to traverse to perform the following tasks: target-searching, boundary-following and obstacle avoidance. The total length of the path traversed determines the efficiency of path traced. This proposed method also enhances the utility of the ACO algorithm by designing and creating a feasible ACO graphical user interface. Further we carried out the research on the working of the ACO algorithm by performing systematic testing, simulations and real-time implementation. **Improvements:** Future work could involve the implementation of a positioning system that allows the robot to determine its actual real world position and then provide feedback to the ACO algorithm so that adjustments could be made. The basic ACO algorithm could be modified to model ants to move in eight directions. All simulations and real time implementations could be done in pre-known environments with static well defined obstacles. By including dynamic obstacle avoidance capabilities, the range of real life applications in which the algorithm could be implemented on would be greatly expanded.

Keywords: ACO Algorithm, Known Environment, Mobile Robot, Machine Learning, Navigation Planning, Static Obstacles

1. Introduction

Ant Colony Optimization is considered as a meta-heuristic algorithm and is popularly used for the solution of complex optimization problems[1]. Inspired by the behavior of actual ants in their quest for food, this concept was proposed[2] who discovered that certain ants tend to sublimate pheromones on the path they traversed during their search for food. This pheromone trail attracts other ants in search for food to follow the same path taken by first traversed ant. Paths which successfully led to food sources had more ants returning to gather more food and thus had a stronger concentration of pheromones[3]. On the other hand, paths that led to dead-ends caused fewer and fewer ants to use that path over time, which subsequently resulted in decreasing concentrations of pheromones due to evaporation and other natural causes. Furthermore, shorter paths tend to have a higher concentration of

*Author for correspondence

pheromone as ants travel back and forth on a short path more frequently as compared to ants on longer paths. Over time, this bias in the pheromone level between successful paths and dead-ends, and between shorter paths and longer paths, leads to ants converging on the shortest and most successful path to their food source.

This paper involves the application of the Ant Colony Optimization (ACO) algorithm onto an autonomous mobile robot guiding it to follow a near optimal path. The idea of using Ant Colony Optimization metaheuristics[4] in robot navigation has been explored in the past by several researchers. Notably, the authors tested the effectiveness of the ACO algorithm when applied to an autonomous mobile robot in a controlled, well defined two-dimensional environment. This was done by conducting simulations on computers in a paper titled "Ant Colony Optimization for the boundary-following robot problem", using user-defined planar simulation environments.

In the study, they represented the simulation environment as an integer matrix with different regions labeled as different integers. They then proceeded to study the variation of the fitness distribution of "Ant paths" over various iterations[5], in rooms of differing sizes. The authors then concluded that the algorithm's performance decreased as room size and complexity increased.

The basis of this paper has its foundation in the research work done previously. It adopts the fundamental concepts and ideas which were applied by the authors in their ACO algorithm. In addition to verifying previous research work, this paper also focuses on extending the work done by the authors to different areas. Firstly, an Ant Colony Optimization algorithm has been written from scratch, but with considerable variations from the original concept and algorithm parameters used by the authors themselves. In particular, algorithm equations and parameters have been generalized and modified to reduce calculation time and to promote algorithm simplicity. New concepts such as negative pheromone deposition, environment-size-dependent equations, loop elimination and a double layer path optimization algorithm have also been included in the amended ACO algorithm.

Furthermore, the functions and capabilities of the algorithm have been extended to support the robot in accomplishing the tasks of target-searching and obstacle avoidance in addition to its basic boundary-following capabilities. This serves to create a more robust and intelligent robot that can compete a wider range of tasks apart from the simple task of following walls, effectively extending its usefulness in many situations. The size of environments in which the robot is able to work in has also been extended 18-fold compared to that of environment.

Merging of algorithms with ACO is possible to give enhanced results. One such is a Hybridizing method combining Ant Colony Optimization and Lorentz transformation which has been used as Chaos Optimization Algorithm with NASA datasets to estimate the software cost[6]. Whereas in another article Bacterial Foraging Algorithm (BFOA), firefly algorithm, Ant Colony Optimization (ACO), bee colony optimization, cuckoo optimization were reviewed to find the optimal result. In that ACO as graph based algorithm suits to find the optimal path in determining the cost of the route compared to other algorithm in discrete environment[7]. There are variations of the ACO techniques they have used to solve problems, representation of problems, transformations, parameters used and advantage of the techniques and used variants[8].

Finally, the capabilities of the algorithm have been tested in real-time on an actual autonomous mobile robot in a real obstacle-filled environment and its performance monitored. The ultimate goal of this paper is to prove the effectiveness and usefulness of the Ant Colony Optimization algorithm in the field of artificial intelligence.

2. Overview of Methodology

The base on which the algorithm works on is a two dimensional array of numbers which represents the environment. These numbers take a value of zero or higher. The higher the number, more pheromones are located in that location, thus the more attractive it is for ants to move to that location. A value of zero signifies the presence of an obstacle.

This two dimensional environment is travelled on by multiple agents known as Ants. Ants are able to move one unit space in the environment in one unit time and can move in one of four main directions at a given step. All ants are allowed to take one step at each time unless the ant has reached the goal or exceeded its maximum number of steps allowed.

2.1 Ant Colony Optimization

Two similar but functionally different algorithms based on the concepts of ACO were written for the purpose of this

paper, one for the purpose of performing target-searching in an obstacle filled environment, the second for the purpose of performing boundary-following in a closed area. The pseudo-codes for both algorithms are presented below. Although mentioned separately, it is to be noted that the obstacle avoidance capabilities of the robot have been merged with the target-searching and boundary-following functions, and thus performed as a single task.

Considering that this algorithm is to be used on environment sizes much larger than that of previous researchers, many of the general ACO algorithm equations used to calculate various parameter values have been simplified and modified to help reduce calculations and simulation time[9]. A few statements and equations to be noted are as follows:

1. For the Target-searching algorithm, the probability for an ant to move to any one of its immediate neighbors in a single ant step is calculated according to the formula:

$$Probability\ to\ move\ to\ X = \frac{Pheromone\ in\ X}{\sum Pheromone\ in\ all\ neighbours} \quad (1)$$

2. For the boundary-following algorithm, the probability for an ant to move to any one of its immediate neighbors in a single ant step is calculated according to the formula:

$$Probability\ to\ move\ to\ X = \frac{\sum Pheromone\ in\ all\ neighbours}{Pheromone\ in\ X} \quad (2)$$

This equation is such that it promotes the exploration of new walls that have not been discovered by ants.

3. The amount of pheromones deposited on a successful path is determined using:

$$Pheromone\ deposited = \left(\frac{1}{N}\right)^{\frac{P}{2}} \quad (3)$$

Where,
N = Number of steps taken along the path and
P = Pheromone deposition constant.

4. Negative pheromones are deposited according to the formula:

$$Phermones = \frac{-0.001}{mapsize} \quad (4)$$

5. Pheromone evaporation conducted at the end of each iteration:

$$R = E \times P \quad (5)$$

Where,
R = Remaining pheromones,
E = Pheromone evaporation constant and
P = Currently available pheromones.

A. Target-searching pseudo-code

```
Target_Searching(robot_curpos, robot_endpos, map_num)
Initialisation:
Initialise_Var(init_pher, colony_size, num_iter, pher_evap, pher_depo, total_antstep, stat_ant, wall_min_dist)
Load_Sim_Map( map_num, init_pher )
Iterations:
        FOR current_iter 1 to num_iter
                WHILE Ants_Still_Moving
                        FOR current_ant 1 to colony_size
                                Calculate_Prob( robot_curpos, robot_nextpos )
                                Move_History<———Position ( robot_curpos )
                                Ant_Ctrl(wall_min_dist, previous_step_backtrack, sqr_loop_backtrack)
                                new_pos <————————— Assign( Calculate_Prob (i, i+1) )
                                move_ants( i )
                                IF Ant_State( current_ant, total_antstep ) > = max_steps
                                        Return Failure
                                ELSE Colony ( current_ant, robot_curpos ( i ) ) == robot_endpos ( i )
                                        Return Success
                                Stop ( ants_moving )
                Pheromone_evap_map( m_i, n_i)<—( 1- ( pher_evap ) ) * pher_depo
                Pheromone_deposit(current_ant)<—sqrt( 1/colony(current_ant, max_steps))^pher_depo
Target_Searching ( )
```

B. Boundary-following pseudocode

```
Boundary_Following( robot_curpos, map_num)
Initialisation:
    Initialise_Var(init_pher, colony_size, num_iter, total_antstep, robot_nextpos, wall_min_dist, wall_max_dist )
        Load_Sim_Map( map_num, init_pher )
Iterations:
    FOR current_iter 1 to num_iter
        WHILE Ants_Still_Moving
            FOR current_ant 1 to colony_size
                robot_nextpos <---- Assign_probability(current_ant )
                Move_History <---- Position ( robot_curpos )
                    Ant_Ctrl( wall_min_dist, wall_max_dist, previous_step_backtrack, sqr_
                        loop_backtrack )
                new_pos <---------- Assign( robot_nextpos ( i ) )
                move_ants( i )
        Pheromone_Deposit(current_ant) <---( colony (current_ant, robot_curpos ) == init_pher )
    IF Pheromone_Deposit( m_i, n_i) == init_pher
        path_blacking_out( Pheromone_Deposit ( m_i, n_i ) )
Boundary_Following ( )
```

2.2 Enhancement of Ant Colony Optimization Algorithm

The full extent of this paper involves not only the design and writing of an Amended Ant Colony Optimization algorithm, but also another two related programs that work on top of the ACO algorithm, which ultimately enables real time implementation of ACO in an autonomous mobile robot. The following Figure 1 flow diagram shows the hierarchy of various algorithms created for this paper.

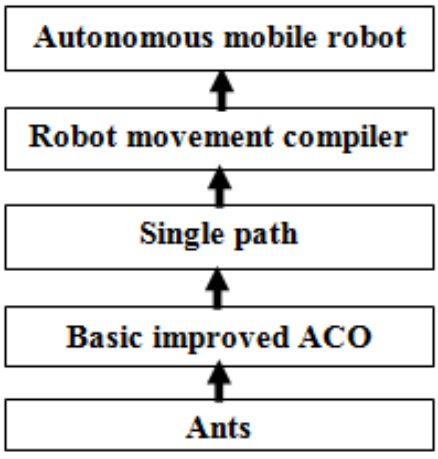

Figure 1. Flow Diagram of ACO

2.2.1 Single Trajectory Selection Algorithm

This algorithm takes as input the pheromone map from either the target-searching or boundary-following function and from it chooses a single best path for the robot to travel on. While seemingly redundant in purpose considering that the ACO algorithm would be able to accomplish the same task, this second algorithm actually helps to reduce the simulation time.

The ACO algorithm is a relatively slow algorithm that requires longer computational time as it employs hundreds of "Ants" over hundreds of iterations. Such long computational times could prove detrimental to the success of its implementation in real time. On the other hand, the Single Trajectory Selection algorithm is a very simple and much faster algorithm, although much of the capabilities available in the ACO algorithm are lost in the tradeoff for speed. With these two options at hand, instead of running the slower ACO algorithm as long as it takes to converge to a single path, our proposed algorithm allows the ACO algorithm to run until a relatively narrow spread of pheromones is found. At this point, the Single Trajectory Selection algorithm would take over the task and perform a much faster and more effective convergence process to a single path.

This Single Trajectory Selection algorithm works in similar ways to the ACO algorithm in that it chooses the next step to take to depending on the amount of pheromones in the four main directions. This is done as shown in the following page.

$$\sum Pheromone\ in\ X\ direction = \sum_{1}^{N} \frac{Pheromone\ M\ units\ away}{M^2} \quad (6)$$

Where,
N is the distance from the robot's current position to the nearest obstacle in X direction and
M = 1, 2, 3 … N

The next step to take would be to move in the direction with the most summed pheromones among the 4 main directions. The above formula is such that it places more importance on its immediate surroundings compared to locations farther away from the robot. This is done so that the robot does not get distracted by less optimal paths that run in parallel with the path it is travelling on, and also by patches of high pheromone locations along unsuccessful paths.

This algorithm differs with the ACO algorithm in that it uses only a single agent, the "virtual robot", in a single set of iterations, to travel along the half-converged pheromone path output by the ACO algorithm. It is because of this simplicity that the algorithm gains its speed.

2.2.2 Mobile Robot Motion Compiler Algorithm

This algorithm functions by taking the output of the Single Trajectory Selection algorithm – a matrix containing all the points along the single selected path, and then compiles this series of coordinates into robot comprehensible movement instructions.

All algorithms have so far been written using Matlab. On the other hand, the mobile robot used for the purpose of this paper, the AmigoBot by Mobile Robots Inc has a compiler that works only in C programming. Therefore an Aria-Matlab adapter layer written by[10] specifically for robots made by Mobile Robots Inc was used to convert Matlab based instructions into C functions. This library allows the programmer to call pre-specified functions using Matlab codes which are then converted to C codes and communicated with the mobile robot.

The Mobile Robot Motion Compiler algorithm is built around this adapter layer and performs its task of compiling robot instructions in two main steps. First by breaking the path matrix into multiple sets of coordinates and calculating the necessary robot orientation and movements needed to be made until it reaches the next set of coordinates[11]. The algorithm then inputs these values into the relevant functions from the adapter layer and communicate them with the robot. The pseudo-code for this algorithm is as shown.

Mobile Robot Motion Compiler Pseudocode

```
ACO_to_Robot( robot_path_log, cur_angle )
Connection Establishment with robot:
    Robot_Start_Position          get_position( robot )
Variable Initialisation:
    Initialise_Var( robot_curpos, robot_nextpos, min_dist, last_pos_matrix, robot_cur_axes, next_angle )
Iterations:
    FOR
        IF [ robot_path_log ] ! = last_pos_matrix
                    Calculate_dist( robot_curpos, robot_nextpos )
                    robot_orient <---- Calculate_orientation( robot, robot_cur_axes )
                    Move_Robot( robot_orient )
                    robot_angle_to_rotate <---- [ next_angle – cur_angle ]
                    Set_Delta_Heading( robot, robot_angle_to_rotate )
                    Check_Sonar_Obstacles( robot, min_dist )
                    Move_Robot( robot, min_dist )
                    robot_curpos <---- shift_var( robot_nextpos )
                    cur_angle <---- shift_var( next_angle )
Ending Connection Establishment with Robot:
    Robot_End_Position      get_position( robot )
    Disconnect ( robot )
ACO_to_Robot ( )
```

A graphical user interface (GUI) specifically designed for the ACO algorithm was created using the Matlab GUI toolbox[12]. The GUI has 18 functions available on it and serves as the platform with which input and output are transferred between the ACO program and the user. A user has to select the simulation map in which the algorithm is to be run and then input the start and end positions of the robot along with the mode the algorithm to perform – target-searching or boundary-following. The GUI would then be able to display the pheromone map travelled on by the ants, the "transition" map on which the algorithm performs calculations, or the "robot map" in which the virtual robot travels on in real time. These features, and more, are specifically designed to assist any researcher in using the algorithm to study its performance or to conduct testing.

3. Experimental Outcomes

Figure 2. Depiction of Pheromone.

With the help of the GUI, testing and simulation are done using the ACO algorithm. Figure 2 shows an example of how the pheromone map would look like at the end of a typical run of the ACO algorithm. A whiter location represents the availability of a larger amount of pheromones in that location and therefore having a higher attractiveness[13] to be travelled on by ants. Figure 3 shows the path travelled on by the robot after the pheromone map has been analyzed by the Single Trajectory Selection algorithm.

This research paper discusses the observations made when varying the major parameters of the Ant Colony Optimization algorithm, namely, size of the colony, steps in iterations, deposition constant of pheromones, and the evaporation constant of pheromones.

3.1 Size of the Colony

Colony Size represents the total number of ants that simultaneously search the simulation environment at each and every iteration of the ACO algorithm. Simulation results show that the larger the colony size, the more paths are found between the start and goal positions. This is in agreement with the theory of Ant Colony Optimization as more ants are available to search all possibilities of the workspace. A larger colony size also means more ants successfully reaching the goal and therefore the availability of much "whiter" paths. Nevertheless, having colony sizes that are too large significantly increases the simulation time without providing better simulation results.

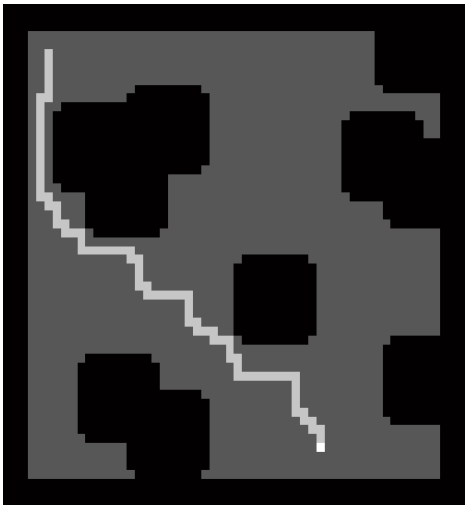

Figure 3. Track traversed by the mobile robot.

3.2 Steps in Iterations

This variable indicates the number of times the colony of ants is to be called upon in series to search the environment in a single complete simulation run. ACO theory gives us the understanding that a larger number of iterations gives more chances for successful paths to be travelled on through the pheromone-biased path selection process and thus solidifying its status as a viable path through pheromone deposition. At the same time it allows less optimum paths found at the beginning of the simulation to fade away through natural pheromone evaporation as explained in ACO theory. Nevertheless, a larger number of iterations would logically mean longer simulation time. The trick is to find the best trade-off between simulation time and path efficiency.

Simulation results further solidify this understanding[14]. A relatively small number of iterations results in multiple paths from start to goal position. This is because ants were still not able to converge to a single optimum path within the limited amount of time (limited iterations). As the number of iterations increased, ants began to converge to a single optimum path (with some slight deviation). The path taken gradually becomes more streamlined and "smooth" compared with the previous simulation runs. Simulation results improve until a point when the effect of pheromone evaporation takes over pheromone deposition and thus solutions to the problem, although still existent, appear very much faded and less obvious.

3.3 Deposition Constant of Pheromones

The rate at which ants deposit pheromones on the path they travelled on after successfully reaching the goal is known as the pheromone deposition constant. Equation 3 shows the relation between the pheromone deposition constant and the amount of pheromones deposited. It is to be noted that since "N" is always greater than 1, therefore the amount of pheromones deposited is inversely related to the pheromone deposition constant.

Simulation results suggest that a low pheromone deposition constant (more pheromones deposited) would increase the chances of ants converging onto a single path. This is because the high amount of pheromones deposited after an ant successfully reaches the goal would result in the path being able to stay on the map for a longer period of time before evaporating away through natural evaporation, thus giving it a higher chance that another ant would discover the same path and reinforce it. On the other hand, this may lead to the convergence to a less optimal path in the final solution, as a less optimal path quickly gets converged onto by ants before a better path can be found. A higher pheromone deposition constant (less pheromone deposited) would reduce the chances of a sub-optimal path being converged upon. This is because any single path would require more ants travelling (and depositing pheromones) on it before it can become prominent in the final solution. Nevertheless, a high pheromone deposition constant may cause a good path to fade away due to natural evaporation before it can be reinforced by other ants.

3.4 Evaporation Constant of Pheromone

This constant determines the uniform rate at which pheromones are taken away from the entire simulation map regardless of any other effects or variations. Although seemingly working in opposition to the core concept of Ant Colony Optimization, i.e. pheromone deposition, pheromone evaporation plays a major role in regulating the growth and development of suitable solutions in the algorithm. It does so by allowing early discovered but non-optimal paths to fade away over time so as not to affect the movements of ants towards a less optimal solution. Pheromone evaporation also limits the growth of found solutions so that yet to be discovered better paths are given the opportunity to be travelled on (pheromones on discovered paths are limited so that the relative chances of ants moving to an yet to be discovered path is higher). In other words, pheromone evaporation encourages ant exploration.

High pheromone evaporation constant has the effect of causing paths to fade away quickly. On one hand, it helps to remove less travelled paths from the pheromone map. On the other, good solutions may be evaporated away if the constant is set too high. Low pheromone evaporation constant would result in a mess of pheromones all over the simulation map as pheromones deposited on non-optimal paths remain on the map for a longer amount of time, thus attracting ants to travel on non-optimal paths.

3.5 Summary of Specifications

One main point to be noted from this Table 1 is that some of the major variables in the ACO algorithm are map size dependent. This gives the entire algorithm the ability to function in simulation environments of varying sizes and design without having to re-program or re-define any part of the algorithm.

Table 1. Specifications

Specifications	Values
Colony Size	10 x map size
Number of iterations	10 x map size
Pheromone deposition constant	0.50
Pheromone evaporation constant	0.01
Negative pheromone deposition rate	-0.001 / map size
Initial pheromone level	0.30

4. Real Time Experimental Performance

4.1 Hardware Equipment Framework

The hardware equipment framework required to implement the ACO algorithm on a mobile robot consists of three main components: A computer with an Ethernet device and capable of running Matlab, a wireless router, and a mobile robot. The computer functions as the brain of the system by hosting the ACO algorithm and all other algorithms and plays the role of performing all necessary calculations based on input from the user or the sensors onboard the mobile robot. The router serves to establish an Ethernet connection between the computer and the mobile robot. For the purpose of this paper, a wireless Ethernet connection was established between the router and mobile robot so that the robot would be able to roam around in the environment without wiring constrains. The mobile robot used for the purpose of this paper is the AmigoBot developed by Mobile Robots Inc.[15].

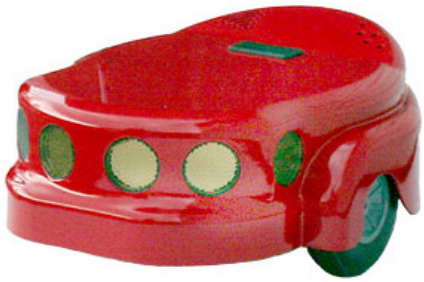

Figure 4. AmigoBot Robot.

Figure 4 shows the AmigoBot robot. It has eight sonar sensors located around the robot which was used to implement the obstacle evasion capabilities of the robot (apart from the obstacle avoiding capabilities implemented in the ACO algorithm)[16]. A wireless serial to Ethernet device was attached to the robot to allow its onboard processor to establish a connection with the computer through the wireless router.

4.2 Mobile Robot Navigation Monitoring

The mobile robot's journey through the obstacle filled environment consists of multiple iterations of a set of movements: a reading of the sonar sensors for the purpose of obstacle evasion, followed by a rotation of the robot, and ends with the robot moving forward to the next step at which the set of movements is repeated again and again until the destination is reached. Such a journey from start to end position is not error free. Two main causes have been identified to result in discrepancies between the robot's simulation calculated position and its actual real world position. A major source of error is due to inaccurate modeling of the real world environment into the simulation model. In this research, modeling of the environment was done by taking measurements of the workspace and all obstacles within it and converting those measurements into map data. Secondly, errors are caused by inaccurate robot motor movements as it rotates and translates the robot throughout the journey. These errors stack up over time, thus causing significant positional errors over longer journeys. In the average journey of about five meters, an error of about 15cm, or 3% error, is obtained at the end of the journey. An error of such magnitude can be neglected in larger environments or in environments with relatively few and simple obstacles. On the other hand, closely placed or tricky obstacle courses requiring sharp turns in the robot's movements could cause the robot to collide with obstacles around it despite having basic obstacle evasion abilities.

5. Further Enhancement

This paper although having met its functional goals would still have to be improved on before any large scale uses could be made out of the discussed achievements. Listed below are a few suggestions for further enhancement:

1. The basic ACO algorithm could be modified to model ants to move in eight directions, each 45 degrees apart instead of the 90 degrees in the four direction model that is currently being used[17]. While the outcome of such an implementation could only be speculated on, it is believed that the new model would allow ants to be more sensitive to changes in direction of movement, thus creating more efficient, smoother paths.
2. The only sensors that allow the robot to locate itself within the real world environment are the sonar sensors located around the robot. Even so, these sensors only allow the robot to detect obstacles within its immediate vicinity and do not determine the robot's actual position in the global environment. Future work could involve the implementation of a positioning system that allows the robot to determine its

actual real world position and then provide feedback to the ACO algorithm so that adjustments could be made. This would greatly reduce the chances of the robot getting lost or colliding with an obstacle in the environment due to undetected errors in its movement.

3. Dynamic obstacle avoidance capabilities are a pre-requisite in most autonomous mobile robot navigation. In this paper, all simulations and real time implementations are done in pre-known environments with static well defined obstacles. By including dynamic obstacle avoidance capabilities, the range of real life applications in which the algorithm could be implemented on would be greatly expanded. Nevertheless, efforts have to be made to implement such capabilities without increasing too much the calculation and simulation time.

6. Conclusion

This paper has dealt with the development of an Amended Ant Colony Optimization algorithm based on a work previously conducted. This ACO algorithm consists of two separate custom designed algorithms. This helps in the implementations of path planning based on ACO to be done in real-time. The main tasks of boundary-following, target-searching and obstacle avoidance have been designed and accomplished and also given an explanation of each and every file and function available in the ACO program and how they interact to create a functional ACO algorithm. Simulations were done using the GUI to study the characteristics of the written algorithm and to determine the best set of algorithm parameters and program structure that serves the work best. The usage of the GUI, simulation results and an analysis of algorithm output were all discussed. The developed algorithm has been proved to be successful in searching for the shortest viable path between two points, although its simulation time and consistency could be improved on. Real time implementation of the algorithm was done using the AmigoBot robot and details were given on both the hardware and software setup in order to perform real time implementation. An analysis of the outcome of implementation was then performed and discussed. The robot is able to navigate its way in the obstacle filled environment based on the path calculated by the ACO algorithm.

7. Acknowledgements

The authors would like to thank the Monash University, Malaysia for permitting us to carrying out the research project. The authors also thank SRM University for providing us facilities to update the work.

8. References

1. Kose M. Ant Colony Optimization for the Boundary-following Robot Problem. Eastern Mediterranean University, Computer Engineering Department. 2004.
2. Goss S, Aron S, Deneubourg JL, Pasteels JM. Self-organised shortcuts in the Argentine Ants, Naturwissenschaften. 1989; 76:579–81.
3. Maniezzo V, Gambardella LM, Luigi F. Ant Colony Optimization. New Optimization Techniques in Engineering.Volume 141 of the series Studies in Fuzziness and Soft Computing. 2004; 101–21.
4. Wikipedia. Metaheuristics. Available from: http://en.wikipedia.org/wiki/Metaheuristic Date accessed: 09/ 2016.
5. Mataric MA. Distributed Model for Mobile Robot Environment-Learning and Navigation. MIT Artificial Intelligence Laboratory Technical Report.1990; AI-TR-1228.
6. Dizaji ZA, Gharehchopogh FS. A Hybrid of Ant Colony Optimization and Chaos Optimization Algorithms Approach for Software Cost Estimation. Indian Journal of Science and Technology. 2015; 8(2):128–33.
7. Kanaka Vardhini K, Sitamahalakshmi T. A Review on Nature-based Swarm Intelligence Optimization Techniques and its Current Research Directions. Indian Journal of Science and Technology. 2016; 9(10):1–13.
8. Sakthipriya N, Kalaipriyan T. Variants of Ant Colony Optimization- A State of an Art. Indian Journal of Science and Technology. 2015; 8(31):1–15.
9. Koza JR. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Massachusetts. 1993.
10. Borgstrom J. ARIA and Matlab Integration With Applications. Master's Thesis Project. Department of Computing Science, Umea University. 2005.
11. Schwab B. AI Game Engine Programming. Charles River Media, inc. 2004.
12. Kose M, Acan A. Knowledge Incorporation into ACO-Based Autonomous Mobile Robot Navigation. Eastern Mediterranean University, Computer Engineering Department. 2004.
13. Cicirello VA, Smith SF. Ant Colony Control for Autonomous Decentralized Shop Floor Routing. The Robotics Institute, Carnegie Mellon University. 2001.

14. Dorigo M, Stutzle T. Ant Colony Optimization. MIT Press, Massachusetts Institute of Technology. 2004.
15. Mobile Robots Inc official website. Available from: http://www.mobilerobots.com/. (Accessed: September 2016)
16. Mobile Robots Inc. Team AmigoBot Operations Manual. 2016.
17. Ross SJ, Daida JM, Doan CM, Bersano-Begey TF, McClain JJ. Variations in Evolution of Subsumption Architectures using Genetic Programming: The Wall Following Robot Revisited. Genetic Programming: Proceedings of the First Annual Conference, The MIT Press, Stanford University, 1996. p. 28–31.