Path planning of coverage region for autonomous robots

Final project AAE 590 - Multi-Agent Systems and Control
Narciso Jesús Soto Picazo
Purdue University

***Abstract*—Path planning for autonomous vehicles has become one of the most intensely researched topics of our time. One of the reasons underlying this phenomenon is the increasing interest in autonomous delivery systems and planetary, exploration among others. The approach to solving a path planning problem varies depending on the end goal of the autonomous system that is being designed. One particularly interesting problem is that of the coverage region. This problem consists in covering a prescribed area in the most *efficient* way possible, which means avoiding visiting the same spots more than once. In addition, this problem must often be combined with obstacle avoidance techniques. Possible applications range from exploratory rovers in other planets, to autonomous indoor vacuum cleaner robots. This work focuses on the latter application, since the limited amount of computational resources on-board these small robots poses an interesting challenge in terms of algorithm optimization, striving for short computation times, great performance, and no need for extensive storing capabilities. In order to achieve this task, a genetic algorithm approach is proposed, which differs from previous attempts in the incorporation of variable chromosome length and non-random chromosome mutation, in order to improve the speed and efficiency of such algorithm.**

***Index Terms*—Path planning, Coverage region, Autonomous Robots, Genetic Algorithm**

I. Introduction and Motivation

While exploratory rovers and delivery systems receive much attention when it comes to path planning research, one kind of application that is often overlooked is that of autonomous vacuum cleaner robots. In moving towards a more autonomous future, these robots can play a role as important as autonomous delivery systems, increasing life quality and allowing mankind to direct their efforts towards those tasks that really do matter. As a consequence of the lack of research, current solutions are, on the one end, cheap but really inefficient, or really effective but also really expensive. These issues have undermined the adoption of this new technology to the general public. One of the reasons that an optimal solution has not been found yet comes from the fact that effective path planning algorithms that do not require a lot of computational power have not being developed, forcing the cheaper options to rely on random walk algorithms that are extremely ineffective (Which were studied in 2018 by T. Edwards and J. Sörme [1]), and the more expensive options to rely on cloud-based solutions, which are costly to maintain and require prior mapping of the room, using the robot itself.

Very recent developments in this area were attained in 2019 by M.A. Yakoubi and M.T. Laskri [2] by making use of genetic algorithms. This work attempts to improve on their method by introducing variable chromosome length and directed mutation that yields faster convergence to optimal paths, therefore making the implementation of this algorithm onto actual systems much easier and effective.

II. Problem Formulation and Main Results

A. The genetic algorithm

Before jumping into the details of the proposed method, let us first introduce the basis of the genetic algorithm. According to [3]:

This algorithm is based in nature, and it relies in random mutation and *survival of the fittest* concepts. The algorithm is based in the evaluation of a randomly-generated population of *chromosomes*, being each chromosome a possible solution to the current path planning problem. The solution expressed in each chromosome is encoded by using a *representation scheme*. Once an initial random population has been created, a *mating pool* is generated, which then gets modified by application of *crossover* and *mutation* operators. This modified mating pool becomes the next population, which gets evaluated in terms of a *value* function, and the best chromosome in each population gets compared to the *best-so-far* chromosome. This method allows to solve optimization problems which are not easily solvable by either means since the value of each solution cannot be defined with a traditional mathematical function, nor can the components of the solution being expressed without the used of a *representation scheme*.

B. Environment discretization

One key aspect of making this path planning algorithm lightweight, so that it can be integrated into actual robots, consists in *discretizing* the environment. This means that, instead of keeping track of every single robot's position, the surrounding area will be divided in discs with the same size as the robot itself, named *cells*, and the motion of the robot will consist in eight primitive steps that will conform whole paths. These 8 steps are those corresponding to the eight positions immediately surrounding the robot.

By applying this method, instead of keeping track of the robot's position, the position of *each visited cell* will be recorded, which will save lots of memory and computation time, since the amount of information that needs to be reviewed at each step will also be greatly reduced.

C. Representation scheme

In order to use a genetic algorithm, a key step is to define a representation scheme, which will later be used to encode the optimal paths information into an array of numbers. As explained in section II-B, since the environment will be divided in *cells* the same size as the robot, the eight basic movements that the robot can make can be expressed in terms of the relative change in the cell's indexes (Analog to coordinates X and Y in a plane). The representation scheme also needs to provide information which will later be used to compute the fitness of each possible path, such as the distance involved in performing each primitive step or the amount of rotation needed in order to perform such a step (Assuming the robot can only move forward and therefore needs to rotate and advance to move in different directions). The proposed representation scheme is shown in Table I

Value	Primitive direction	Required distance [m]	Angle wr. to N [rad]	ΔX	ΔY
1	N (Up)	D	0	0	D
2	NE (Up-right)	$D\sqrt{2}$	$\pi/4$	D	D
3	E (Right)	D	$\pi/2$	D	0
4	SE (Down-right)	$D\sqrt{2}$	$3\pi/4$	D	$-D$
5	S (Down)	D	pi	0	$-D$
6	SW (Down-left)	$D\sqrt{2}$	$-3\pi/4$	$-D$	$-D$
7	W (Left)	D	$-\pi/2$	$-D$	0
8	NW (Up-left)	$D\sqrt{2}$	$-\pi/4$	$-D$	D

TABLE I: Representation scheme

Notice that in Table I, D represents the robot's diameter, and each number in the *value* column will be used in a chromosome to represent that particular primitive motion. Therefore, a chromosome represents a path composed of several primitive motions, each of them represented by a number as described in Table I.

D. Variable chromosome length

One of the main improvements of the presented algorithm over previous methods, especially when compared to [2], is the fact that in this method a variable chromosome length has been considered. In [2] a fixed chromosome length was used, which resulted in mini paths that would get the robot, at most, to the limit of its sensing range. However, this presents some problems, as in that research, the robot was assumed to be able to *see* through obstacles. Therefore, planning a mini path around an obstacle was not an issue since the robot had all the information of the environment around itself despite of the obstacle.

Real world sensing systems, often based in laser-ranging or ultrasonic technologies, do not have the ability to *see* or perceive the environment beyond existing obstacles. In this scenario, planning paths that go beyond the robot's distance to the closest obstacle does not make sense. In particular, a predicted mini path may go around an obstacle, assuming there is available space behind it, only to find that this is not the case while performing the maneuver, having wasted computational resources in calculating a longer path that cannot be executed. The algorithm developed in this work, utilizes a chromosome length that advances the robot, at most, a distance described by:

$$d = min\{d_{\text{Closest obstacle}}, s_{\text{range}}\} \tag{1}$$

E. Mating pool selection

Following the guidelines in [3], the mating pool selection process needs to ensure that the best chromosomes from the initial population go into the next one. In order to achieve this, a process know as *tournament scheme* was used.

This process consists in choosing two random chromosomes from the initial population, then evaluating the fitness of both of them, and only allowing the best one to pass into the mating pool. The mating pool is a set of chromosomes that will be modified by *crossover* and *mutation* operators to develop the following population, as described in the following sections.

F. Crossover and directed mutation

Crossover and *mutation* operators are two distinct kind of operations that can be applied to the population to (originally) promote change and randomness.

The *crossover* operator, as implemented in this work, follows its original purpose of generating randomness in the population. A *single parent* crossover operator was chosen for simplicity. It consists in switching the position of two randomly chosen elements inside the same chromosome. Traditionally, the *crossover* operator takes sections from a chromosome and exchanges it with a different chromosome. However, single parent crossover was chosen not only because of its increased simplicity and speed, but because the *mutation* operator was implemented in a non-random manner, so extreme changes to the chromosomes are not desired.

The *mutation* operator, traditionally, randomly changes elements inside the chromosome. However, in an attempt to increase the speed of convergence of this algorithm to an optimal solution, the *mutation* operator uses a more deterministic approach, described as follows.

The value function programmed in this algorithm, as it will be further explained in II-G, automatically sets the value of a chromosome as 0 if such chromosome results in the robot running into an obstacle. Under those circumstances, the value function also returns the index of the primitive motion that resulted in such a crash. The mutation operator, in this context, operates by changing such primitive motion by a different one. In case a chromosome yields a feasible route, then the mutation operator randomly changes one element of the chromosome.

In addition to these operators, *elitism* was also applied. This kind of operation consists in replacing the worst chromosome in every generation by the best-to date chromosome, therefore ensuring that each generation has at least one chromosome that is *at least*, as good as the best solution found to that moment.

G. The value function

The main indicator of how effective is a path planning algorithm for a coverage region, is an *efficiency* described as follows:

$$\text{Val} = \frac{\#\text{ of cells visited}}{\text{Required time}} \tag{2}$$

However, this expression of the value function alone does not yield satisfactory results. The underlying reason is that this value function favors *straight* trajectories over any other trajectory, which leads to intersecting paths that leave many sections of unvisited cells surrounded by visited cells. In order to solve this, and increase the *compactness* of the resulting paths, a *cohesive* factor was designed.

The cohesive factor works by substracting from the total time of a primitive motion, the time it takes to rotate, *if and only if* the resulting rotation yields a path that runs parallel and adjacent to previously visited cells *or* obstacles. This operation promotes the generation of *zig-zagging* patterns that enable the robot to visit first the cells that are closer to itself, then moving on to the rest.

It must be noted that, in case a chromosome results in a crash with an obstacle, the value of such chromosome is automatically set to zero, and the index of the element is returned as part of the output of the value function so that that element can be targeted specifically by the *mutation* operator. Additionally, if the robot is surrounded only by cells that has visited already, any possible mini path will yield as a result a value of 0. In such a situation, the robot starts performing a random walk, until a cell that has not previously visited is within range.

Finally, it must also be noted that this algorithm only uses information available to the robot's sensors at each time, together with a history of visited cells. It does not record every single robot position, making it faster to read previously visited positions, diminishing the decision time.

H. Results and comparison with random walk algorithms

This algorithm was put to the test with 3 different randomly generated environments, which are depicted in Figures 1, 2 and 3. Please note that the black circles represent obstacles, while the white circles with red outlines represent non-visited cells. In addition, these same environments were simulated with a random walk algorithm, which is widely used in this kind of application, and compared to the results obtained with the genetic algorithm.

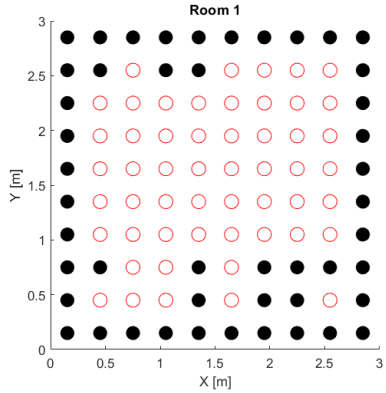

Fig. 1: Environment 1

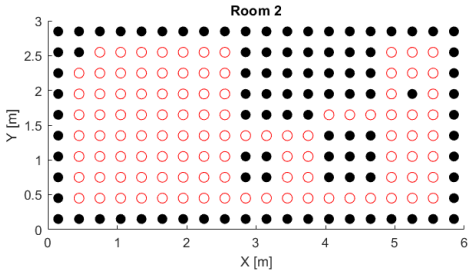

Fig. 2: Environment 2

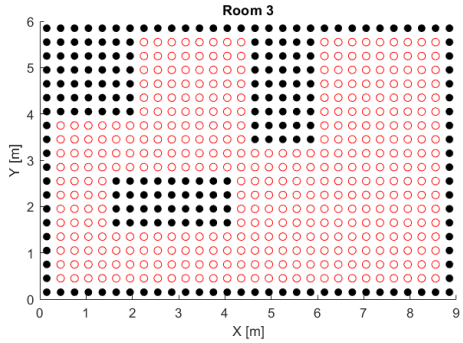

Fig. 3: Environment 3

the characteristics of these 3 environments are defined in Table II.

	Environment 1	Environment 2	Environment 3
Size [m]	3×3	3×6	6×9
Obstacle density $[m^{-2}]$	10%	20%	25%

TABLE II: Environment characteristics

The conditions for the simulation were the following:

- Robot's linear speed: $0.2\ [m/s]$

- Robot's angular speed: 0.3 $[rad/s]$
- Robot's shape: Round
- Robot's diameter: 0.3 $[m]$
- Sensing range: 10 times D

Together with these parameters, stopping conditions were defined for the simulation, not only for the GA, but for the random walk algorithm too. The stopping criteria is as follows

- Genetic algorithm:
  - Visit at least 90% of empty cells
- Random walk:
  - Visit at least 90% of empty cells
    -or-
  - Simulation time reaches 3 times the total time needed by the genetic algorithm

The results for the 3 different environments are depicted in Table III, comparing the results with the random walk algorithm. Moreover, the specific evolution of the visited cells over time and the amount of repeated cells visited over time for Environment 2 are represented in 4, both for the genetic algorithm and the random walk algorithm.

		% Area covered	Total time [s]	Redundant area $[m^2]$
Environment 1	Gen. Algo.	92.5	141.3	1.71
	Random Walk	90.6	412.22	13.41
Environment 2	Gen. Algo.	93.3	181.3	0.09
	Random	67.3	544.02	17.28
Environment 3	Gen. Algo.	90.8	643.8	1.44
	Random Walk	51.662	1932.6	77.4

TABLE III: Summarized results for all environments

As it can be seen in Figure 4, the progression of the amount of cells covered for the fist time is practically linear in time for the genetic algorithm, meaning that almost every cell visited was visited for the first time. This same tendency was also observed for Environments 2 and 3, showcasing the effectiveness of the algorithm. Moreover, the genetic algorithm yields few if any cells visited multiple times, most of them arising at the last stages of the simulation, as the robot becomes surrounded by no cells that has not been visited before, and its forced to perform a random walk until it finds an area never covered before.

On the other hand, the random walk algorithm starts performing similarly to the genetic algorithm. However its effectiveness decays rapidly as more cells in the environment become visited, therefore visiting many cells multiple times, as it records no notion of which cells have been visited before This tendency became even worse in larger environments with wide open spaces. The random walk algorithm was not capable of covering 90% of the room in excess of three times the amount it took to the genetic algorithm.

It must be noted that this satisfactory results were obtained by just using the data available to the robot's sensors at each instant of time and a list of previously visited cells.

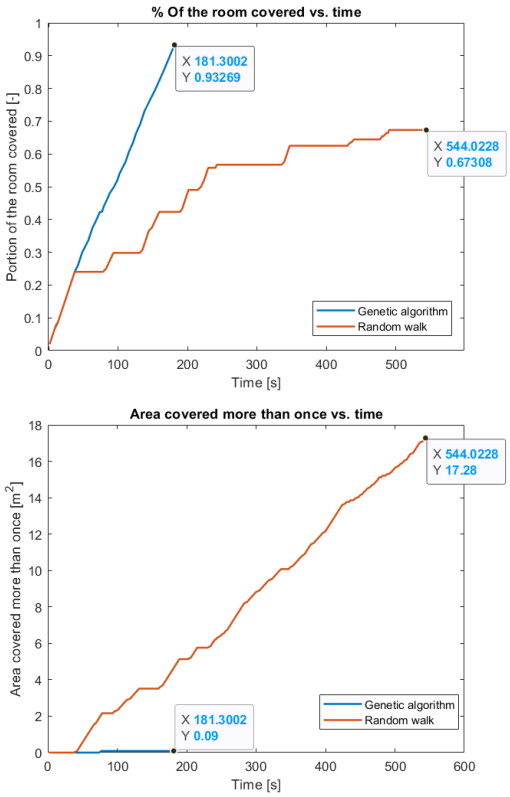

Fig. 4: Time history for environment two of: Percentage of cells covered (Upper) redundant area covered (Lower)

III. Your Ideas of Further Improvements

The results obtained with the algorithm designed in this work were extremely satisfactory. However, it was tested using computer simulations, and the performance in real physical systems remains untested. One common issue with these kinds of algorithms is that the robot's position needs to be known. When subject to noise and external disturbances, keeping track of the robot's steps is not an accurate solution to determine its position, and position estimators need to be applied.
Moreover, in order to speed the algorithm even more, to make its implementation into small micro controller boards, the value function can be substituted by a lookup table where all possible combinations of primitive paths are already recorded, reducing the time required for computations.

References

[1] T. Edwards and J. Sörme. A comparison of path planning algorithms for robotic vacuum cleaners. *Thesis research project, KTH Sweden*, 2018.
[2] M.A. Yakoubi and M.T. Laskri. The path planning of cleaner robot for coverage region using genetic algorithms. *Journal of Innovation in Digital Ecosystems*, 3:37–43, 2019.
[3] Edwin K.P. Chong and Stanislaw H. Zak. *An Introduction to optimization*. Wiley, 4 edition, 2013.