sciendo

DOI 10.2478/ama-2018-0024

acta mechanica et automatica, vol.12 no.2 (2018)

GENETIC ALGORITHM FOR MOBILE ROBOT ROUTE PLANNING WITH OBSTACLE AVOIDANCE

Konrad K. KWAŚNIEWSKI*, Zdzisław GOSIEWSKI*

*Faculty of Mechanical Engineering, Bialystok University of Technology, ul. Wiejska 45 C, 15-351 Bialystok, Poland

k.kwasniewski@doktoranci.pb.edu.pl, z.gosiewski@pb.edu.pl

received 31 July 2017, revised 18 June 2018, accepted 21 June 2018

Abstract: Nowadays many public and private institutions begin space studies projects. Among many problems to solve there is a planet exploration. Now rovers are controlled directly from the Earth, e.g. Opportunity. Missions must be planned on the Earth using simulators. Much better will be when the mission planner could set the target area and work to do and the rover will perform it independently. The solution is to make it autonomous. Without need of external path planning the rover can cover a much longer distance. To make autonomous rovers real it is necessary to implement a target leaded obstacle avoidance algorithm. Solutions based on graph algorithms use a lot of computing power. The others use intelligent methods such as neural networks or fuzzy logic but their efficiency in a very complex environment is quite low. This work presents an obstacle avoidance algorithm which uses the genetic path finding algorithm. The actual version is based on the 2D map which is built by the robot and the 2nd degree B-spline is used for the path model. The performance in the most cases is high using only one processor thread. The GA can be also easily multithreaded. Another feature of the algorithm is that, due to the GA random nature, the chosen path can differ each time on the same map. The paper shows the results of the simulation tests. The maps have the various complexity levels. On every map one hundred tests were carried out. The algorithm brought the robot to the target successfully in the majority of runs.

Key words: Space Robotics, Mapping, Genetic Algorithm, Obstacle Avoidance

1. INTRODUCTION

The exploration of other planets, besides Earth, is a great challenge of the aeronautics. Nowadays, planetary rovers/robots are commanded and controlled from Earth. The long distances between celestial objects cause long delays in data transfer, so the real time control is impossible. Moreover, the mission headquarter can communicate with the rover only during the proper planets conjunction. It is inconvenient and makes many troubles. The future lies in the rover autonomy. Autonomous mobile robots can make better use of the time and solve problems when they occur unexpectedly.

The most significant system that the autonomous rover should have is one which allows it to move on the planet surface. Multitude types of rover designs provide many different possibilities of movement. However, for every kind of them there exist obstacles that are impassable. This makes path planning and obstacle avoidance systems absolutely necessary.

The simplest way to obtain an obstacle avoidance method is to build a map of environment, mark the obstacles and use one of the path finding graph algorithms, such as Dijkstra algorithm, A* (Kumar et al., 2010), Bellman-Ford algorithm and the others. Unfortunately, the graph algorithms require big computational power.

Another way to avoid obstacles is use of Artificial Neural Networks (ANNs), fuzzy controllers (Berisha et al., 2016; Jincong et al., 2009; Chen and Juang, 2009), neural-fuzzy systems (Raulcezar and Carlos, 2016; Cheol-Joong and Dongkyung, 2015), ant colony optimization algorithm (Zhi-Qiang and Zi-Xing, 2006).

There are also computational methods that recognize obstacles displacement and compute the robot trajectory by remaining the obstacles at the distance (Zong et al., 2006; Ping et al., 2009; Peng et al., 2015; Langisetty et al., 2013). The potential fields method (Cerqueira et al., 2016) is successfully used also.

Algorithms based on genetic algorithms (GA) used for path finding and path planning (Hua et al., 2008; Hu and Zhu, 2010; Burchardt and Salomon, 2006; Alajlan et al., 2013; Mathias and Ragusa, 2016) are the latest and very important navigation algorithms. This kind of methods has a great development potential. Thanks to them it is possible mark out the correct global and local path without significant computation time cost. However, their performance highly depends on their design details.

Other types of methods are those strictly based on type of sensor. As an example, can be mentioned a SLAM (Simultaneous localization and mapping) method (Chen L. et al., 2007) that uses for mapping only monocular camera, what simplifies the system and reduces its cost. Another interesting way to perform obstacle avoidance is grid mapping based on Q-Trees (Sencan O, Temeltas H, 2018). The idea of grid mapping is also used in a method presented in this paper but in a different implementation way.

In the paper we propose a path finding genetic algorithm based on obstacle avoidance method, in which a 2nd degree B-spline for path marking is used. The GA is specially designed for high convergence that allows to obtain results very fast in comparison with classic graph algorithms. It is particularly important in a robot which must work under real time regime and save the control energy.

DOI 10.2478/ama-2018-0024

2. TARGET SYSTEM AND ENVIRONMENT MODEL

This version of the algorithm is designed for a mobile wheeled autonomous robot which is able to turn around its middle axis and its dimensions allow to turn around (360°) completely inside the given square field with safety margin. The robot moves between middle points of the neighbouring fields. The aim of the robot is to find a path between start and target fields as fast as possible. The target environment is mostly flat area filled with obstacles in various shapes. It can be both a simple hall with chairs and a complex maze-like building interior.

The environment is approximated to the 2-dimensional rectangle map of square fields. The map is represented as 2D array where fields have weights in the range $[0,1]$. The value 0 (black colour on map) means that field belongs to the desired path field, whereas 1 (white colour) is the value of the field with an obstacle. All values between 0 and 1 mean the cost of going through them.

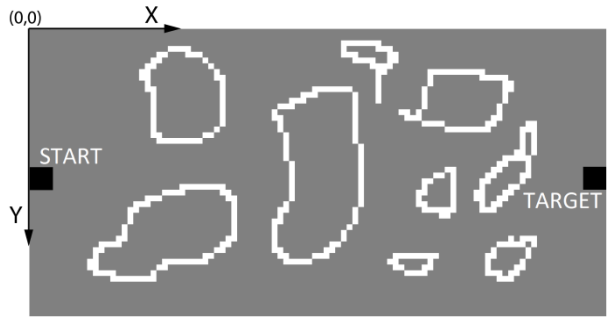

Fig. 1. Example of the map

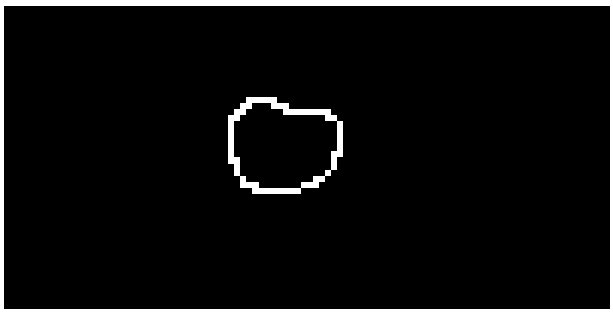

Fig. 2. Test map 1

Fig. 3. Test map 2

This way of the environment representation provides many possibilities of influence over the path shape. Fields with lower values are more likely to be visited by the robot than those with the higher ones. This makes possible to determine the preferred path. The algorithm presented in this paper uses a value 0.5 (grey colour) as the map base value. In this case the robot can look for a path freely. Example of the map is shown in Fig. 1. Dimension of the tested maps is 100x50 fields.

Fig. 4. Test map 3

Fig. 5. Test map 4

The axes are also shown in Fig. 1. All the maps are oriented in the same way, so in the rest of them the axes are omitted. The start (0,24) and target (100,24) fields are marked.

The known apriori maps used for tests of the GA and Dijkstra algorithms are shown in Figs. 2 – 5. The unknown apriori maps for path finding with help of obstacle avoidance method are shown in Figs. 6 – 9.

3. GENETIC ALGORITHM FOR PATH FINDING

The aim of the genetic algorithm (GA) is to find the fastest (in sense of calculation and motion time) path from field occupied by robot to the target in the unknown environment.

3.1. The path

The way of the path repesentation is an important part of the algorithm. In our method a 2nd degree B-spline is used for it. Using description from (Piegl, 1997) a 2nd degree B-spline curve is defined by the following equation:

$$B(u) = \sum_{i=0}^{n} N_{i,2}(u) P_i \quad 0 \le u \le 1 \tag{1}$$

where: P_i – the control points, $N_{i,2}(u)$ – the 2nd degree B-spline basis functions defined on the nonperiodic knot vector U of $m+1$ knots.

$$U = \{0,0,0,u_3,\dots,u_{m-3},1,1,1\} \tag{2}$$

The number of knot points (m) is connected with number of the control points and describes the distances between control

points in the measure of the B-spline which whole length is always 1 and it is its reference frame. The knot points in our algorithm are equidistant.

As can be seen, a vector of the control points (i.e. P_i) is completely enough for the exact B-spline description. So, the genetic algorithm can optimize the path using chromosomes that contain a control points coordinates. Fig. 14 shows an example of the B-spline path.

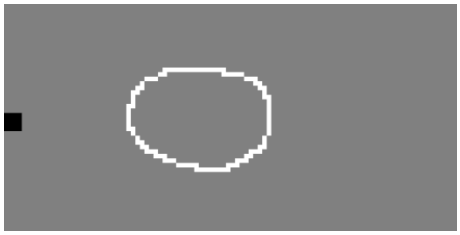

Fig. 6. Map 1

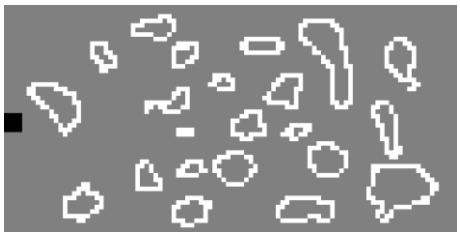

Fig. 7. Map 2

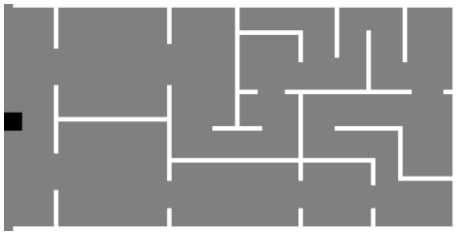

Fig. 8. Map 3

Fig. 9. Map 4

The number of the control points is set at the algorithm start and remains constant until the termination condition is satisfied. The number of control points should be chosen experimentally. There is no control points adaptation mechanism in the algorithm at this stage.

The B-spline curves are not a main subject of the paper. For detailed description please look at (Piegl, 1997).

3.2. Chromosome representation

A natural numbers vector is used as a chromosome to represent the path structure. Every pair of genes (x and y) represent coordinates of the exactly one control point of the path. General equation is presented below:

$$C = [x_1, y_1, \dots, x_i, y_i, \dots, x_k, y_k] \quad 1 \le i \le k \tag{3}$$

where: $\mathrm{x_i}$ – *i*-th *x* coordinate, $\mathrm{y_i}$ – *i*-th *y* coordinate, k – number of the control points.

Following equation shows an example of the chromosome:

$$\mathrm{C_e} = [0,0,3,3,1,5] \tag{4}$$

As can be seen from eq. 4, a chromosome $\mathrm{C_e}$ describes a path with three control points. The first is $(0,0)$, the second $(3,3)$ and the last one is $(1,5)$.

3.3. Initial population

The initial population is divided into three parts. The first one is generated as a straight line from the position of the robot to the target point. Then it is divided into segments. The number of segments depends on the control points number that is set in advance, i.e. for three control points the line is divided into two segments. Finally coordinates of every point between the first and the last points can be modified by adding with a fixed probability a random value which can be positive or negative. This part provides a range of possible solutions for the case in which the path can be marked out in the neighbourhood area of start and end fields.

The way of the generation of the second part of the initial population is quite different. All control points without the first one and the target are chosen randomly from the neighbourhood of the start point. It provides a material for crossing to solve situations in which the robot has the obstacles very close to itself.

The control points of the third part are chosen randomly from all the map with a specific margin. Thanks to that the algorithm is fed with points that accelerate finding the path in the case in which it is impossible to mark out the short path.

In our case the total initial population consists of 30 chromosomes. Every part of population is of the same size - 10 chromosomes. Due to that the path finding has much better performance that in the case of completely randomly generated population.

At this moment it is important to mention that all clearly unmentioned parameters such as range of a neighbourhood in the second part of population or size of the margin in the third part can have a high influence on the performance. Futhermore quality of chosen parameters can change with increasing or decreasing of the map dimensions. The selection of the parameters is very important to obtain a well-optimized algorithm for the robot environment.

3.4. Fitness function

As was described in the section 2, the fields in the map can have various values:

- fields of the target and start area – 0;
- fields of the obstacles – 1;
- fields open for the robot – all between 0 and 1.

The fitness function of the chromosome is the sum of the fields values which belong to the path. The value 1 is changed to the very large number (VLN; in comparison to the scale of other field values and the sum of all of them, e.g. 100000). It is necessary to recognize easily if the path contains fields ocuppied by

DOI 10.2478/ama-2018-0024

obstacles or not. The path is recognized as incorrect if it contains at least one field which is occupied by an obstacle. So, the incorrect path fitness is always greater than VLN. Moreover, the points of the path that are situated outside the map are treated like fields with obstacle.

The fitness function design has 2 steps:

1. B-spline conversion from control points (chromosome) to waypoints.
2. Summing up the map fileds values pointed by waypoints (described by following equation).

$$F_i = \sum_j M_{P_{i,j}} \tag{5}$$

where: $\mathrm{F_i}$ is the *i*-th individual fitness value, M – is the map of environment, $\mathrm{P_{i,j}}$ is the *j*-th waypoint of the *i*-th individual waypoints array.

First, the B-spline knot points are computed using an algorithm A3.1 from (Piegl, 1997). The knot vector (U) is prepared to obtain 5-way points of the curve. The number of points is small for a better performance. Their coordinates are approximated to the nearest centers of the map fields. Next step is a conversion of the knot points to the continuous curve. It is done by connecting the knot points by a straight line. Then they have to be rasterised. The DDA (Digital Differential Analyzer) algorithm is used for this. Ccomputed coordinates of all the points of the path make it possible to determine the path fitness. The values of the map fields that belong to the path are added up, where the value 1 is replaced by the VLN.

3.5. Selection

The algorithm uses a rank selection method. Firstly, all chromosomes are sorted ascending where the sort key is the fitness value. The crossing is done for the best chromosomes (1/4 of the population). They are copied to the new population.

3.6. Crossing

Parents of the new chromosomes are chosen randomly from the selected population. For every pair an arithmetical crossing is proceeded which is specified as follows:

$$C_{new} = \begin{cases} D \cdot C_A + (1-D) \cdot C_B, & fit(C_A) < fit(C_B) \\ D \cdot C_B + (1-D) \cdot C_B, & fit(C_A) \geq fit(C_B) \end{cases} \tag{6}$$

where: $\mathrm{D} = 0.8$ – parents mixing coefficient, $\mathrm{C_A}$, $\mathrm{C_B}$ – chromosomes of parents, respectively, $\cdot$ – element-wise multiplication, $\mathrm{fit}()$– function that returns chromosome fitness.

3.7. Mutation

For every gene of the chromosome obtained from crossing, excluding the two first and the two last genes, a random value from $[-\mathrm{h}/2; \mathrm{h}/2]$ (where h is the lower of the map dimensions) with probability of 40% is added.

3.8. Termination condition

The execution of the algorithm stops under three conditions:

– after obtaining a path with fitness lower than VLN;
– after reaching the given maximal number of epochs;
– independently from the others, termination will happen only after reaching the given minimal number of epochs.

The maximal number of epochs depends on the place in the path finding algorithm, where the genetic algorithm is executed, to obtain the best performance. The minimal number of epochs condition is added to optimize the path in simple cases.

3.9. Method efficiency

A good way of test the method effectiveness is to compare it with an another widely–known method. As a reference the Dijkstra algorithm (Cormen, 2001) was chosen. The comparison of such methods is difficult, because of their different nature. Dijkstra algorithm is a graph algorithm and the proposed GA is a random-based algorithm, whence it follows that results may vary significantly depending on the input data. To compare them a computing time was chosen. It is obvious that it depends on the computational platform performance. To make the results comparative all the tests were executed using one computational platform, which remained unchanged during testing. The implementation of both algorithm procedures was done in Python 3.6.

3.10. Methodology

Four maps were prepared with increasing complexity level (Figs. 2 – 5). Dijkstra algorithm was executed for all maps in the same way. Proposed GA was executed for all maps with three different settings that will be described further. The time of every execution was measured. All the tests were done using the same computational platform to make them comparable. The resolution of maps was 100x50 which gives a graph with 5000 nodes.

3.11. Results

The results are presented in Tab. 1. Important parameters of the GA: minimal number of epochs (min.), maximum number of epochs (max.), population size (pop.), number of control points (kr.).

Tab. 1. Computation times (sec.)

MAP	Dijkstra	GA 1	GA 2	GA 3
Map 1	43.06	0.03	0.07	0.08
	46.73	0.03	0.08	0.08
	44.02	0.03	0.07	0.08
Map 2	42.51	0.21	0.07	0.15
	42.37	0.10	0.14	0.15
	42.24	0.18	0.14	0.15
Map 3	42.29	1.10	3.42	0.59
	42.73	0.77	2.10	0.59
	47.78	N.F.	0.77	1.37
Map 4	41.29	N.F.	N.F.	N.F.
	39.54	N.F.	N.F.	N.F.
	40.06	N.F.	N.F.	N.F.

Explanations:
- GA 1 – min. 1, max. 500, pop. 40, kr. 4;
- GA 2 – min. 1, max. 500, pop. 100, kr. 4;
- GA 3 – min. 1, max. 500, pop. 100, kr. 5;
- N.F. – the path was Not Found.

3.12. Conclusion

As can be seen in Tab.1 the Dijkstra algorithm in every case has found the path and, in every case, it takes about 40 seconds. The GA is much more effective in the simpler maps. Execution times varies depending on GA parameters, but every time it is smaller than 4 seconds. It shows that in that case the proposed GA is about 10 times faster than the Dijkstra algorithm. However, due to GA random nature, it is possible that GA can fail in path finding, like in the 3rd test in map 3. The most complex map has been too difficult for the GA under tested parameter sets. None of the tests succeeded. This implies that the GA is much better than Dijkstra algorithm for path finding in low-complex maps, but it fails in high-complex ones.

In the obstacle avoidance problem in unknown or changing environment, the map is frequently updated likewise the robot path (due to recently found obstacles during exploration). The obstacles of undiscovered area are treated like they do not exist, i.e. the robot treats the undiscoverd area as free from obstacles. This simplifies the environment for path finding. As was shown, the GA is much better than Dijkstra algorithm in this kind of cases, so it is better for purpose which is considered in the paper.

Although the GA is faster, it is useless in the complex cases. If this situation happens, the classical graph algorithm like Dijkstra can be used. Another solution is a return mode which is described in further section.

4. OBSTACLE AVOIDANCE ALGORITM

Whole obstacle aviodance algorithm is divided into parts:
- main loop (Fig. 10);
- step function (Figs 11 – 14).

The step function is the main decision-making part for doing a next movement. It cosists of three parts:
- the return mode (part A, Fig. 12);
- the normal exploration mode (part B, Fig. 13);
- the new path search (part C, Fig. 14).

In this section, three terms are used that describe the paths: current path, past path and a new/result path. The current path is a collection of following fields (their coordinates) which the robot actually uses for the determination of movement direction. The past path is a collection of fields which the robot has visited already (stored in the reaching order). The new/result path is a collection of fields generated as the result of the GA execution. It can be correct (does not contain fields of map that are marked as obstacle-containing; more simply it is collision-free path) or incorrect (otherwise). Its correction and fitness are frequently used in the conditions that follow the GA execution (see Figs. 12-14).

4.1. Main loop

The main loop (Fig. 10) is simple. Robot movement from one field to another is called ‘step’ and the function that is executed after reaching a new field is called ‘step function’.

4.2. Main part of the step function

Step function begins with two conditions (Fig. 11). The first one checks if the return mode is activated or not. If so, the operations of return (part A, Fig. 12) are executed. Otherwise, the correction of the current path (i.e. there is no obstacles on the previously determined path) is checked. If it is true, the operations of moving to the next field (part B, Fig. 13) are executed. If not, the process of searching for the new correct path begins (part C, Fig. 14).

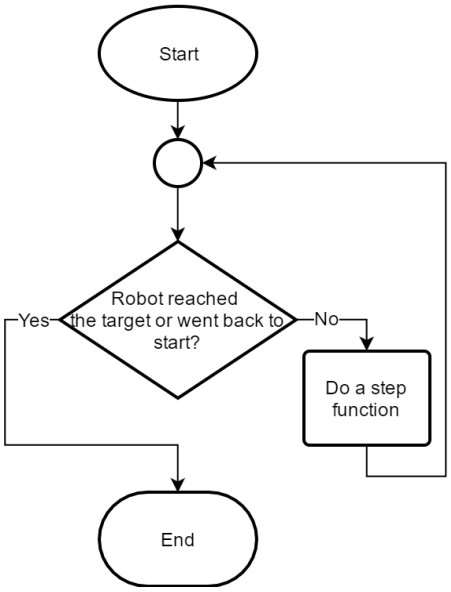

Fig. 10. Main loop flowchart in obstacle avoidance algorithm

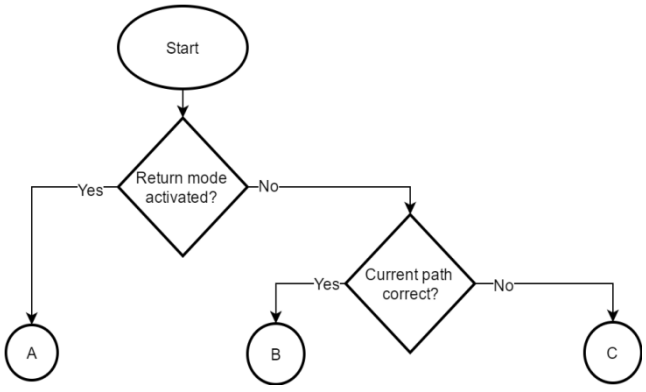

Fig. 11. Flowchart of the algorithm main part

4.3. Part A of the step function

Part A represents the return mode. It is based on idea of coming back in the footsteps and highly increases robustness of the algorithm in environments with multiple obstacles and their complex displacements. The path for robot movement is the past path. A path fields counter starts from the last field (n), so a $\mathrm{n}-\mathrm{i}-1$ field is considered as the “next field”, where i is the number of steps in the return mode.

The return mode operations go as follows: on every second or

DOI 10.2478/ama-2018-0024

third (randomly chosen) passed field of the past path, the GA is executed to search for a correct path. If the result is incorrect then the robot continues its motion along the past path. It also happens in the steps without GA execution (the found path is still incorrect).

If the result path generated by GA is correct then the result path is set as the current path. The return mode becomes deactivated and robot goes to the next field.

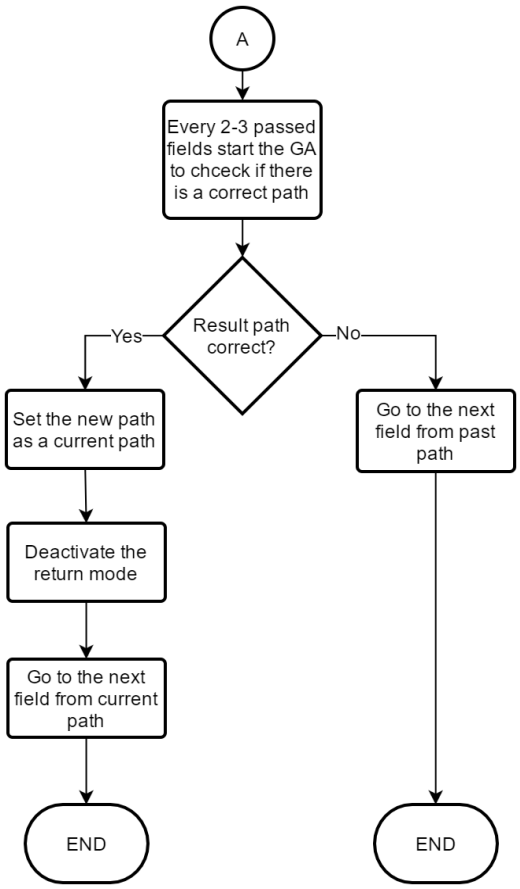

Fig. 12. Flowchart of part A of the step function

4.4. Part B of the step function

Part B represents the normal exploration mode. Its structure is similar to the part A. The robot goes along the current path. Every 5 passed fields it checks for the better path. The GA is configured to do calculations quickly, so the result paths can be quite far from optimal one. The knowledge of obstacles distribution also changes during motion. If the GA cannot find the better path than a current one (incorrect paths are included in this category) then robot moves to next field from the current path. Otherwise the current path is replaced with the result path and robot continues movement along the new path.

4.5. Part C of the step function

The last part – C – is executed when an obstacle is detected on the current path. Then the GA searches for a new path. Here the GA is configured for better path-finding quality. It may take more time of computation than in parts A and B. However, it includes the searching for paths with bigger number of control points (more complex). When the GA found a new correct path then it replaces the current path and the robot moves to the next field. Otherwise the past path is simplified (it is described in Section 5.6) and the return mode is activated. After that the execution of the step function moves to the part A (the return mode).

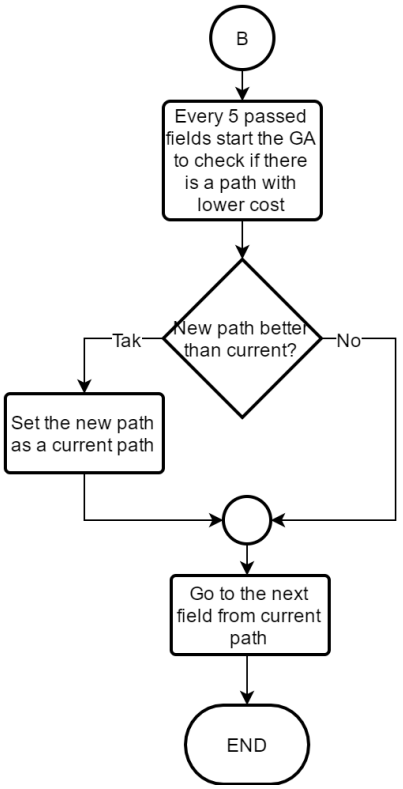

Fig. 13. Flowchart of part B of the step function

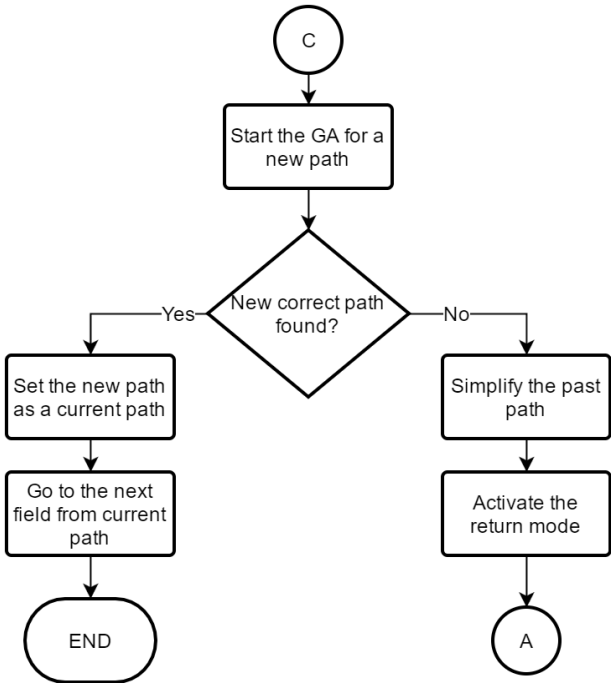

Fig. 14. Flowchart of part C of the step function

4.6. The past path simplification

The past path simplification decreases the length of the path for the robot. Its idea is based on the fact that raw past path can be complicated – it can have many loops. The loops are often in areas there is only one way to get in and out (blind alleys). This means that going to those areas is useless.

Simplification is done by a simple algorithm. Points are checked one-by-one from the first to the last one. For every point it is checked if in the rest of path (i.e. to the last point) there exists a point that has the same coordinates like that one, which is being checked. If a point like that exists, the part of path between that point and point that is being checked is removed and this procedure starts at next point on the list. When the last point has been checked, the past path is successfully simplified.

5. SIMULATION METHOD

The dimensions of the robot area of view are constant both during every test and during all tests. Area is modeled as a isosceles triangle where its top is located in the ocuppied by the robot field. Example of area is shown in Fig. 15. The number of every layer fields is computed with formula:

$$layer(x) = 2x + 1 \tag{7}$$

where: x is number of a layer and can belong to [0, ∞] respecting map dimensions.

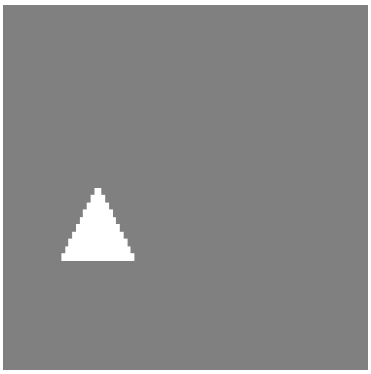

Fig. 15. Example of the simulated robot filed of view – 10 layers; the robot is located at the top of triangle

The algorithm was tested for four maps with different level of complexity. Map resolution was 100x50. 100 tests were conducted on every map. For each test, start point had coordinates $(0,24)$, whereas target point was given by coordinates $(98,24)$ (on the maps marked as a black field – enlarged for clarity). To avoid too long computing time, which is possible because of random nature of the algorithm, a number limit of steps was used. Every map has in description with information about the step number limit size and limit number of the overruns. Robot seeing range was set to 4 layers.

The selection of evaluation criteria was the main problem during the tests. The simple one that the robot achieved the target or not was insufficient. To present performance character a computing time was chosen. For that reason, to preserve comparability, all the tests were executed on a one unchanged computing platform. The results are presented as computing times histograms with added a number of the tests in which the maximum number of the algorithm steps was exceeded. In each map a robot path example is presented.

6. RESULTS

6.1. Map 1 – one simple obstacle

The map 1 can be seen in Fig. 6. Fig. 16 shows an example path. The computing times histogram is presented in Fig. 17. This map contains a single obstacle.

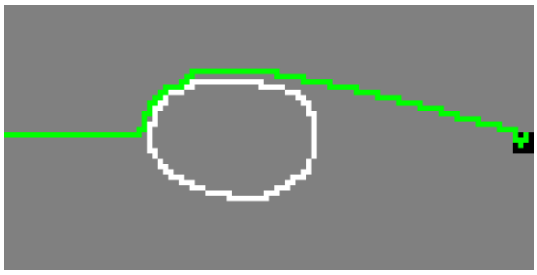

Fig. 16. Map 1 path example

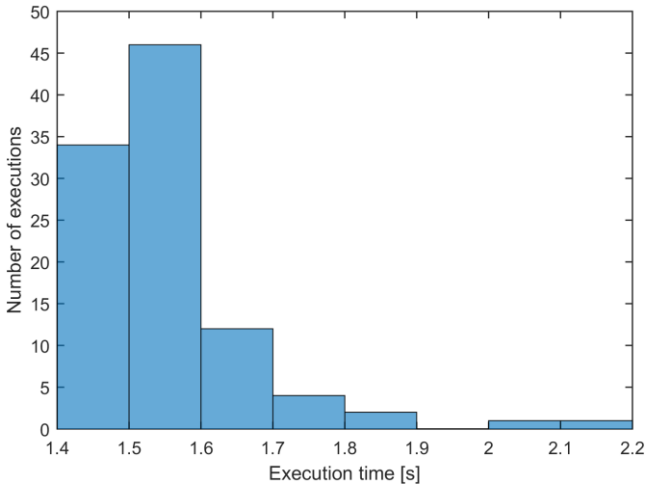

Fig. 17. Map 1 – times of tests histogram

6.2. Map 2 – many simple obstacles

The map 1 can be seen in Fig. 7. Fig. 18 shows an example path. The computing times histogram is presented in Fig. 19. The second map contains 22 simple obstacles placed randomly.

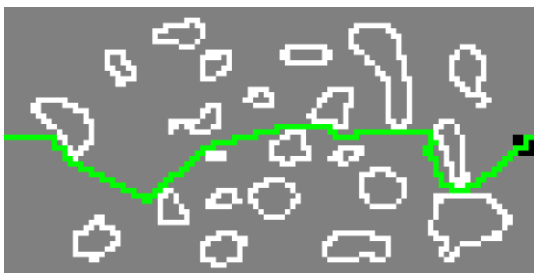

Fig. 18. Map 2 path example

DOI 10.2478/ama-2018-0024

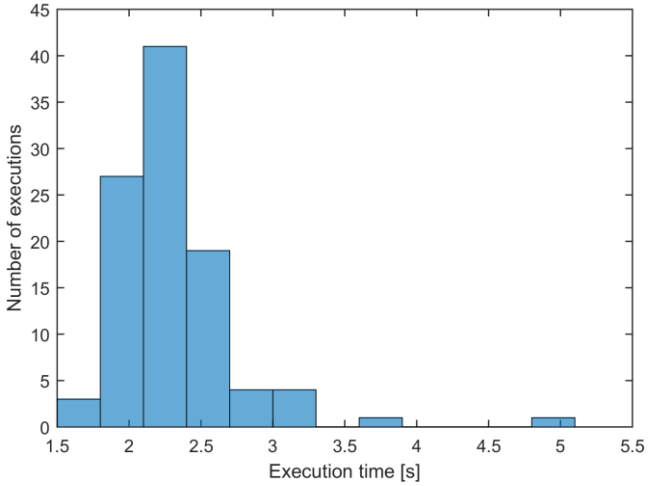

Fig. 19. Map 2 – times of tests histogram

6.3. Map 3 – rooms

The map 1 can be seen in Fig. 8. Fig. 20 shows an example path. The computing times histogram is presented in Fig. 21. The third map represents a case of building interior environment. On the histogram (Fig. 21) two results are not shown – 232 s. and 189 s. because of high difference in comparison to the rest, which would make the histogram illegible.

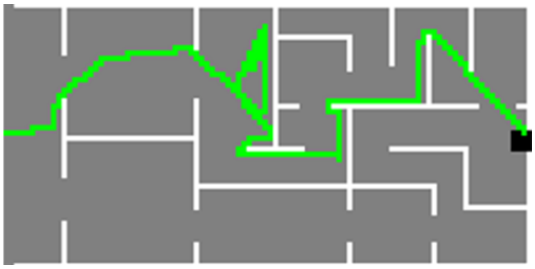

Fig. 20. Map 3 path example

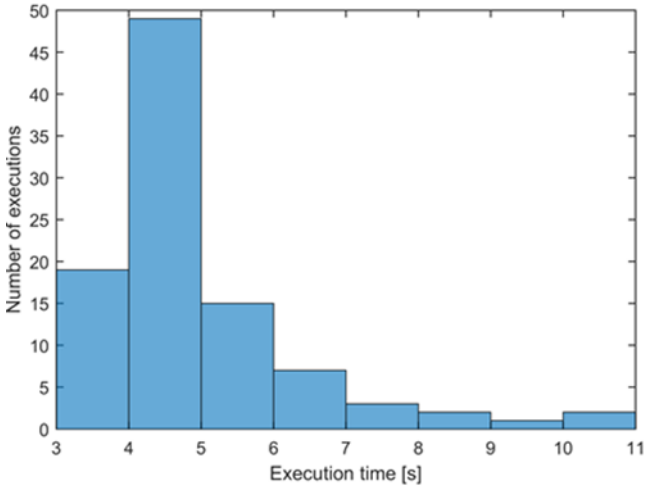

Fig. 21. Map 3 – times of tests histogram (two results not included – 232 s. and 189 s.)

6.4. Map 4 – complex environment

The map 1 can be seen in Fig. 9. Fig. 22 shows an example path. The computing times histogram is presented in Fig. 23.

This map represents an environment with obstacles of different shapes situaded randomly.

Steps number limit: 2000; steps number limit overruns: 0.

Fig. 22. Map 4 path example

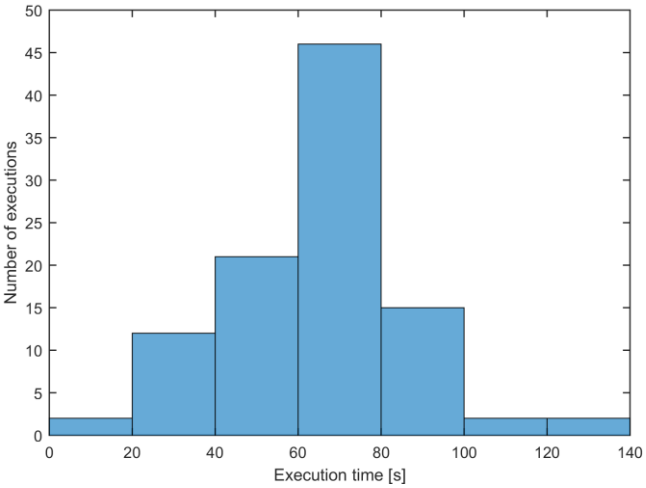

Fig. 23. Map 4 – times of tests histogram

7. CONCLUSION AND FUTURE WORKS

In this paper a method was proposed to solve a fundamental problem in mobile robotics – obstacle avoidance in unknown environment. In all tested maps the robot reached the target in almost all tests under described termination conditions. Every type of obstacles shapes was avoided, even those very irregular and complicated.

A return operation was introduced to the main algorithm. It makes possible for a robot to solve complex maps with 'blind alleys' (areas with only one way to get in and get out) which normally can take much computation time to find a correct path using a genetic pathfinding algorithm. It also ensures that, if there is no possibility to get into the target area (i.e. obstacles completely block the access), the robot will return to the place from where it started.

The GA for pathfinding has specific features. It is designed for fast result obtaining in particular. Even though the algorithm performance is high due to parameters and submethods choice, it makes the restarting of the algorithm necessary. The important problem, which is now solved in basic way, is control points num-

ber selection. The low number provides fast computing but can not describe very complex paths. On the other hand, a lot of control points can determine correct paths for even very difficult cases, but requires more computation time, which is unwanted in robot with real time regime.

The main disadvantage of proposed method it is its complexity. It makes the implementation quite difficult because there are two algorithms to implement (i.e. GA and higher-level decision part).

The algorithm is designed for robots with holonomic constraints. However, with some minor modifications (inter alia additional radius limit fitness, fixed start orientation), it can be used for the non-holonomic ones also.

The next step will be implementation of the algorithm on mobile robot and test of it in the real environment.

The algorithm can be expanded for finding paths in 3D maps. It can be used in robots able to passing some kinds of obstacles (i.e. rovers, humanoidal robots) or even in 3D printers to avoid collisions between printhead and printed elements. This will be the next step after the hardware implementation.

REFERENCES

1. **Alajlan M., Koubaa A., Chaari I., Bennaceur H., Ammar A.** (2013), Global Path Planning for Mobile Robots in Large-Scale Grid Environments using Genetic Algorithms*, International Conference on Individual and Collective Behaviors in Robotics*, El Mouradi Palace 5 Zone Rouristique El Kantaoui Sousse, Tunisia
2. **Berisha J., Shala A., Bajrami X., Likaj R.** (2016), Application of Fuzzy Logic Controller for Obstacle Detection and Avoidance on Real Autonomous Mobile Robot, *5th Mediterranean Conference on Embedded Computing MECO 2016*, Bar, Montenegro
3. **Burchardt H., Salomon R.** (2006), Implementation of Path Planning using Genetic Algorithms on Mobile Robots, *2006 IEEE Congress on Evolutionary Computation*, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
4. **Cerqueira T.A., Santos T. L. M., Conceicao A. G. S**. (2016), A new approach based in potential fields with obstacles avoidance for mobile robots, *XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium*, Recife, Pernambuco, Brazil
5. **Chen Y-S., Juang J-G.** (2009), Intelligent Obstacle Avoidance Control Strategy for Wheeled Mobile Robot, *ICROS-SICE International Joint Conference*, Fukuoka International Congress Center, Japan
6. **Cheol-Joong K. Dongkyoung C.** (2015), Obstacle Avoidance Method for Wheeled Mobile Robots Using Interval Type-2 Fuzzy Neural Network, *IEEE Transactions on Fuzzy Systems*, 23(3).
7. **Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.** (2001), *"Section 24.3: Dijkstra's algorithm". Introduction to Algorithms* (Second ed.). MIT Press and McGraw–Hill.
8. **Hu J., Zhu Q.** (2010), Multi-objective Mobile Robot Path Planning Based on Improved Genetic Algorithm, *International Conference on Intelligent Computation Technology and Automation*, TBD Changsha, China
9. **Hua Z., Manlu L., Ran L., Tianlian H.** (2008), Path Planning of Robot in Three-dimensional Grid Environment based on Genetic Algorithms, *Proceedings of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China
10. **Jincong Y., Xiuping Z., Zhengyuan N., Quanzhen H.** (2009), Intelligent Robot Obstacle Avoidance System Based on Fuzzy Control, *The 1st International Conference on Information Science and Engineering (ICISE2009)*, TBD Nanjing, China
11. **Kumar Das P., Konar A., Laishram R.** (2010), Path Planning of Mobile Robot in Unknown Environment, *Special Issue of IJCCT for International Conference [ACCTA-10]*, 1(2), 3–5.
12. **Lagisetty R., Philip K., Padhi R., Bhat M.S.** (2013), Object Detection and Obstacle Avoidance for Mobile Robot using Stereo Camera, *IEEE International Conference on Control Applications (CCA,) Part of IEEE Multi-Conference on Systems and Control*, Hyderabad, India,
13. **Mathias H.D., Ragusa V.E.** (2016), An Empirical Study of Crossover and Mass Extinction in a Genetic Algorithm for Pathfinding in a Continuous Environment, *IEEE Congress on Evolutionary Computation (CEC)*, Vanouver, Canada
14. **Peng Y., Qu D., Zhong Y., Xie S., Gu J.** (2015), The Obstacle Detection and Obstacle Avoidance Algorithm Based on 2-D Lidar, *Proceeding of the IEEE International Conference on Information and Automation Lijiang*, China.
15. **Piegl L., Tiller W.** (1997), *The NURBS Book*, Springer
16. **Ping G., Wei B., Li X., Luo X.** (2009), Real Time Obstacle Avoidance for Redundant Robot, *Proceedings ofthe IEEE International Conference on Mechatronics and Automation*, Changchun, China.
17. **R. M. F. Alves, C. R. Lopes** (2016), Obstacle avoidance for mobile robots: a Hybrid Intelligent System based on Fuzzy Logic and Artificial Neural Network, *IEEE International Conference on Fuzzy Systems (FUZZ)*, Vancouver, Canada
18. **Zhi-Qiang W. E. N., Zi-Xing C. A. I.** (2006) , Global path planning approach based on ant colony optimization algorithm, *Journal of Central South University of Technology*, 13(6), 707–712
19. **Zong G., Deng L., Wang W.** (2006), A Method for Robustness Improvement of Robot Obstacle Avoidance Algorithm, Proceedings of the IEEE International Conference on Robotics and Biomimetics December 17 - 20, Kunming, China
20. **Chen L., Tang W., John N. W.** (2007), MonoSLAM: Reat-Time Single Camera SLAM, *IEEE Transactions on Pattern Analysis and Machine Intelligece*
21. **Sencan O., Temeltas H.** (2018), A quantized approach for occupancy grids for autonomous vehicles: Q-Trees, *Advanced Robotics*