Article

Time Minimisation of Rescue Action realized by an Autonomous Vehicle

Gosiewski Zdzisław[1*], Konrad K. Kwaśniewski[2]

[1]Bialystok University of Technology, Automatics and Robotics Department; z.gosiewski@pb.edu.pb

[2] Bialystok University of Technology, Automatics and Robotics Department; konrad.krzysztof.kwaśniewski@gmail.com

* Correspondence: z.gosiewski@pb.edu.pb; Tel.: +48-606-483601

Abstract: In the rescue operations the full time of action plays important role. It is a sum of planning, travel, and manipulation (in the action place) phases times. The time minimization of first two phases by autonomous vehicle for remote action is considered in the paper. For known a priori map the path planning consists of local optimal decision collected next in the general algorithm of the optimal path. Such approach significantly reduces time of path planning. The robot features and known sparse obstacles reduce the allowable robot speeds. The time of travel is calculated from allowable velocity profile. So, it can be used to estimate the travel performance. Genetic algorithm and random search-based methods for path finding with travel time optimization are exploited and compared in the paper. All the proposed time optimisation solutions of rescue operation are checked during computer simulations and results of simulation are presented.

Keywords: robotics, autonomy, obstacle avoidance, path finding, path optimization, genetic algorithm, random search

1. Introduction

Mobile robotics is a still growing field of scientific research. Every year it brings new challenges and areas in which mobile robots can help people in dangerous or just simple tasks. These could be a rescue patrol (e.g. searching for earthquake or avalanche victims, people who get lost in a rough terrain), goods transport, medicine transport, patrol in big areas. All of those tasks are often performed in a rough terrain and the fast movement is frequently needed. Therefore, the robots not only have to be able to move fast mechanically, but also algorithmically. It is obvious, that time of travel highly depends on the shape of path, which is chosen.

Many methods are developed to solve the path finding problem. One of the most known is the A* algorithm [1]. It needs the map to be represented as a graph. The result path is discrete and has to be processed to be useful for rover-like robots. Other proposed methods in the literature use artificial neural networks [5, 6]. Their main disadvantages are teaching process, modelling problem so they are difficult to use for global path optimization. Other optimization methods are also frequently used for path finding. Genetic algorithms [3, 12, 13, 14, 19], particle swarm optimization [4, 16] ant colony optimization [7] can be indicated as examples. Some popular methods used for path finding are fuzzy logic [9] potential fields [18], and combining these both [10, 17]. The methods mentioned above often looks for the shortest path or are strictly focused on the avoidance of obstacles. As the shortest path is not always the fastest one, the problem of finding path under shortest travel time criterion remains an interesting field for the research, which importance can rise with increasing global usage of mobile robots.

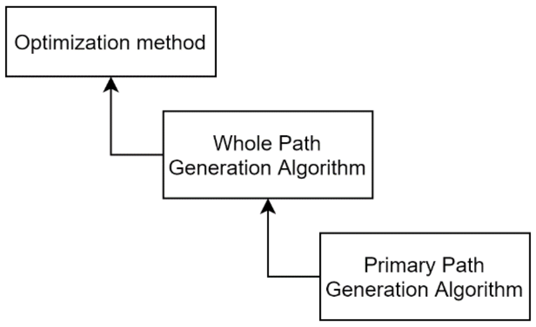
Optimization method
Whole Path Generation Algorithm
Primary Path Generation Algorithm

Figure 1: General algorithms hierarchy.

The aim of the research is to develop the fast method to obtain travel time-optimized paths in a priori known environment or in an environment previously mapped by the robot itself. Two path finding methods are presented in the paper. The first one is based on genetic algorithm and the second one on random search algorithm. Each method is described in two versions: with path fitness estimation and with travel time obtained from velocity profile generator. Both methods are designed in a specified hierarchy, which is presented in Figure 1. It resulted from the analysis of effects obtained in the previous our work [3] which showed limitations of the methods that operates directly on the path shape. The main limitation is a constant complexity of the path (i.e. constant control points number). To overcome that inconveniences another approach was chosen: the optimization method does not operate on path parameters itself, but on the parameters of a whole path generation algorithm (WPGA). The WPGA creates the path from parts of primary paths that are chosen as locally optimal. An important thing about the primary path generation algorithm (PPGA) is that the path is chosen by algorithm from a discrete set of paths. This reduction of a solution space improves significantly performance of the method.

The reason of making a comparison between genetic algorithm and random search algorithm is a result of fact that the previously proposed GA-based method (Kwasniewski and Gosiewski, 2020) has very long times of execution. The previous GA was used only for the estimation of fitness. Since the velocity profile generator has been invented, it brought an interesting opportunity to compare effectiveness of the methods based on the fitness estimation and approximated travel time. Moreover, the comparison shows also, how well the proposed fitness estimation method describes the quality of the path under travel time condition.

Another improvement described in the paper is the robot velocity reduction nearby the obstacles. As a part of the velocity profile generator, it is based on the heuristic knowledge of humanity gained through the ages, that slower motion allows the more precise movement. It is important in the obstacle avoidance problem, because even movement along a proper path can lead to collision, if the robot not follows the path precisely enough. Summarizing, the aims of the research are as follows:

- find short execution time on embedded computers,
- realize travel time optimization for robot global path operating in the area which map is known a priori.

2. Map Model

The algorithm is executed as soon as the environment map was obtained. Due to long time of execution, the work on a priori known map is acceptable only. It is unfit for real time use. The map is modelled as a binary bitmap with terrain approximated as a flat space (Figure 2). The raster of the map is marked by red grid lines. In the maps presented in the paper it will be hidden for clarity as well as the axes. The dark pixels are considered as obstacle-free and the bright ones as inaccessible for a robot. It is assumed that size of one pixel is big enough to be able to contain the whole robot with a safety margin. However, the raster maps have a big disadvantage – their size is usually constant. To overcome it, fields with obstacles are firstly saved as a list of points. Then it is rasterized. It allows to the map size to be adjusted to the all obstacles. Of course, the point list could be used without rasterization, although the collision detection in this case needs to check whole obstacles list for every pixel of rasterized path or to do the geometrical collision detection for every path segment. The collision detection on raster map needs much less tests. It affects performance significantly.

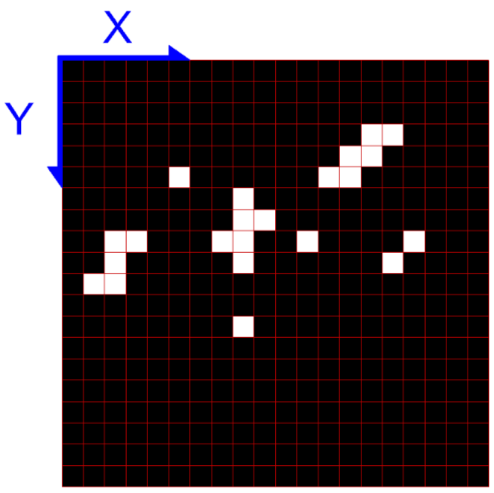

Figure 2: Raster map model. Black fields are obstacle-free, white ones contain obstacles.

Other ways of map modelling are not used since only the raster map gives a fast way of collision detection. Even though it needs the path to be rasterized, which is time-consuming, it is still more effective than e.g. obstacle list, because in that case every segment of the path has to be tested for intersection with all the obstacles. This makes it very inefficient for long complex paths. The 3D map models are unnecessary in the considered case as the problem can be reduced to 2D problem.

Someone could ask, how to obtain the map of the area in the case of e.g. earthquake, because it can make big changes in the environment that makes the previously obtained maps expired. The answer is that it can be provided by UAVs or aeroplanes. The robot can actualize also the map it has got saved in the memory. However, as long as this problem is not the aim of the paper, this question will not be discussed further.

3. Genetic algorithm for path finding

The genetic algorithm for path finding was presented in [20] and its execution time was long. The main feature of the now proposed algorithm is that it does not work on the trajectory shape directly but through the path generation algorithm. It is more efficient way due to large solution space when the GA modify the path control point coordinates. Longer paths create longer chromosomes and expand greatly a solution space, while the path generation algorithm reduces chromosome size to just few values (dependently on the chosen method of a path generation). Another advantage of this approach is no limitation of a length of the path. If a chromosome contains control points coordinates their number cannot be increased.

Generally, the proposed method can be described in the form seen in Figure 3.

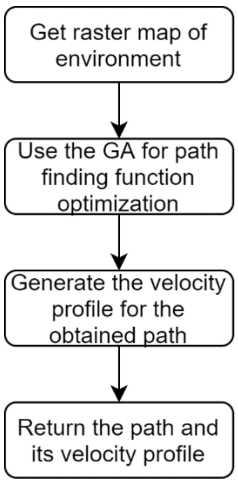
Get raster map of environment
Use the GA for path finding function optimization
Generate the velocity profile for the obtained path
Return the path and its velocity profile

Figure 3: General algorithm flowchart.

3.1. Properties of the GA

The GA for path finding has to, apart from good convergence, return result as fast as possible. To achieve this, the following shape of the GA was chosen:

- Chromosome of floating-point numbers,
- Arithmetical crossing,
- Small population: 30 individuals,
- Roulette selection,
- Elitism: 6 best individuals remain after each epoch,
- Termination after given number of epochs,
- Minimization of the fitness values.

For path generation task the whole path generation algorithm is used. To reduce the parameters number, the lists of parameters are generated. It is described further in section 6.2.

3.2. Chromosome

A chromosome consists of three values: maximal radius, maximal angle range and maximal search point number. They describe a set of parameters for WPGA (whole path generation algorithm). For details please look in section 6.3.

4. Random search algorithm for path finding

During tests of the GA pathfinding method it has been noticed that continuously changing parameters do not result in similar paths, i.e. the function represented by whole path generation algorithm (see section 6.3) is discontinuous. Thus, the GA can have problem to achieve satisfying results, especially in acceptable time. So, the random search method that can be used to solve problems, even in discontinuous spaces, was tested.

The random search algorithm (Figure 4) is simple. At first step, the solution parameters are chosen randomly. Then the fitness of the solution is calculated. Those parameters and fitness value are saved as current ones. At second step the process of getting parameters randomly for a new set and fitness calculation for it is done again. If the new solution has better fitness than the current one, it is set as a current. The last step is repeated until the given number of epochs will be satisfied. Then the current solution is returned as a result.

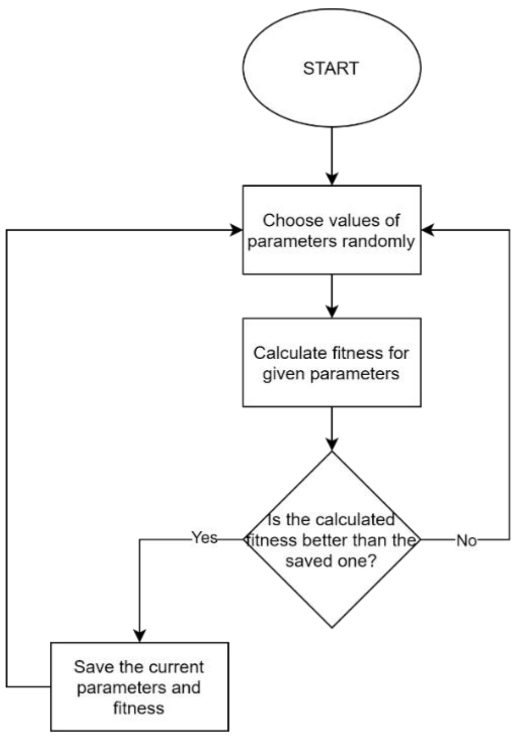
START
Choose values of parameters randomly
Calculate fitness for given parameters
Is the calculated fitness better than the saved one?
Yes
No
Save the current parameters and fitness

Figure 4: The random search algorithm.

As it was in the case of the GA, the whole path generation algorithm is used for path generation task. The number of parameters is reduced by using of the parameter generator. Solutions of the random search algorithm are described in the same way as chromosomes in the GA. Three parameters (maximal radius, maximal angle range and maximal point number) are used by the parameters generator to get parameters set suitable for the WPGA.

5. Path model

The path processing is the most part of computation. A proper model of a path is fundamental to obtain good robot performance. In the literature various models are in use. The most popular are polylines of straight segments, b-splines and NURBS curves. The first one is fast to compute, but before setting it in a robot it needs to be smoothed. On the other hand, b-splines and NURBS curves provides smooth paths. However, they are much more computationally complex and some parameters like radius have to be computed in the form of discrete set with some given resolution. For our application a model is chosen that is free from these disadvantages. It is a polyline, built of arc and straight-line segments, later referred for clarity as arcline (as long as a straight segment can be considered as an arc of infinite radius).

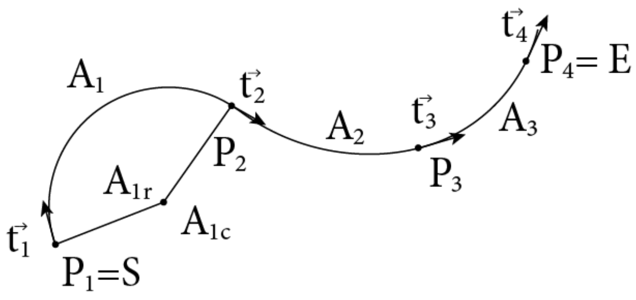

Figure 5: The arcline model.

The arcline can be defined as a list of control points $P_1, P_2, \ldots$ and an initial tangent vector t_1. All of the tangent vectors of the arcline are unit vectors. The A_n arc is always tangential to the A_{n-1} arc at the point P_n.

5.1. Arcline computation

The arcline is computed by calculating sequent segments. Every segment is entirely described by three parameters (Fig. 5):

- Start point P_i,
- End point P_{i+1},
- Initial tangent vector $\vec{t_i}$.

Let's call the i-th segment as A_i. Then the segment can be defined by formula:

$$A_i = \text{segment}(P_i, P_{i+1}, \vec{t_i}) \qquad (1)$$

for $1 \leq i \leq n-1$ where n is a number of arcline control points. Thus, the arcline L is the list of following segments:

$$L = [A_0, A_1, \ldots, A_{n-1}] \qquad (2)$$

Before computation of each segment, it needs to be check, whether the segment belongs to straight or to arc segment category. It is done by comparison of two unitized vectors: tangent vector (Eq. 4) and vector of start and end point (Eq. 3). For A_i segment they are described as follows:

$$\overrightarrow{t_{p1p2}} = \text{normalize}(\overrightarrow{P_i P_{i+1}}) \qquad (3)$$

$$\overrightarrow{t_{tn}} = \text{normalize}(\vec{t_i}) \qquad (4)$$

If those two vectors are equal, i.e. their corresponding coordinates are equal, the segment is straight. Otherwise it is an arc segment.

5.2. *Straight segment computation*

Computation of a straight segment is simple. The only property that can be computed is its length (Eq. 5). The tangent vector at the end point is equal to the initial tangent vector of the segment (Eq. 6)..

$$A_l = |P_i P_{i+1}| \tag{5}$$
$$\overrightarrow{t_{i+1}} = \overrightarrow{t_i} \tag{6}$$

5.3. *Arc segment computation*

The arc segment (Fig. 6) parameters that can be calculated are:

- Arc radius A_r (Eq. 9),
- Arc length A_l (Eq. 11),
- Arc angle β (Eq. 10),
- Arc circle centre point A_C (Eq. 12),
- End tangent vector $\overrightarrow{t_{i+1}}$ (Eq. 8).

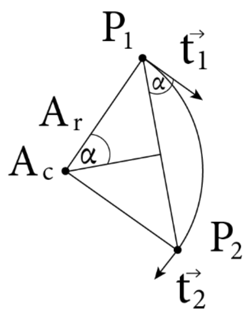

Figure 6: The i-th arc model.

They are described by following formulae:

$$\alpha = \text{angle\_signed}\left(\overrightarrow{t_{P_i P_{i+1}}}, \overrightarrow{t_{tn}}\right) \tag{7}$$
$$\overrightarrow{t_{i+1}} = \text{rotate}(\overrightarrow{t_{tn}}, 2\alpha) \tag{8}$$
$$A_r = \text{abs}\left(\frac{|P_i P_{i+1}|}{2\sin\alpha}\right) \tag{9}$$
$$\beta = 2\alpha \tag{10}$$
$$A_l = \text{abs}(\beta A_r) \tag{11}$$
$$A_C = P_i + A_r \text{ normalize}\left(\text{rotate}\left(\overrightarrow{t_i}, \frac{\pi}{2}\text{sign}(\alpha)\right)\right) \tag{12}$$

where the $\text{normalize}(\vec{t})$ is a function that unitizes the $\vec{t}$ vector, $\text{rotate}(\vec{t}, \alpha)$ is a function that rotates the $\vec{t}$ vector by the α angle, the $\text{abs}(a)$ function returns the absolute value of a, $\text{sign}(a)$

returns the sign of the a and $\text{angle\_signed}(\overrightarrow{t_1}, \overrightarrow{t_2})$ returns the angle with a sign between vectors $\overrightarrow{t_1}$ and $\overrightarrow{t_2}$.

The angle β of the arc is a dubled angle between vectors $\overrightarrow{t_i}$ and $\overrightarrow{P_i P_{i+1}}$. The sign of the angle β defines the side, on which the arc is built. The end tangent vector $\overrightarrow{t_{i+1}}$ is obtained by rotation of the start tangent vector by angle β.

6. Path generation

To make the process of path optimization faster, the optimization methods should not work on the path shape itself. High path complexity expands the solution space significantly, what decreases algorithm efficiency. The solution for that problem is to design an algorithm which sequentially generates a path based on just few parameters. This approach reduces the solution space and allows to operate on the paths of different complexity.

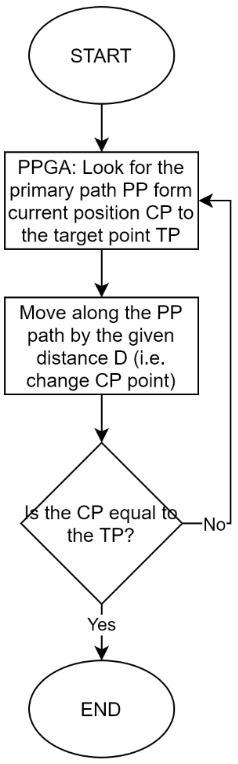
START
PPGA: Look for the primary path PP form current position CP to the target point TP
Move along the PP path by the given distance D (i.e. change CP point)
Is the CP equal to the TP?
No
Yes
END

Figure 7: The Whole Path Generation (WPGA) algorithm flowchart.

The idea is to look for the locally best short path, then move along it some distance and, if it is not a target point, repeat the process. The path generation algorithm consists of two parts: primary path generation algorithm (PPGA) and whole path generation algorithm (WPGA). The first part returns the local path and the second one uses the PPGA to make a path from start to end point. The WPGA flowchart is presented in Figure 7.

6.1. Collision detection

An important part of the path generation is detection of collisions. As the map model is a raster map, a collision occurs when the rasterized path has at least one common pixel with obstacle pixels of the map.

6.2. Primary path generation algorithm – local path

There exist infinity paths that can be built in some area (actually, it is limited by hardware capabilities). As was shown in [20] the process of finding path in that wide solution space can take a long time. However, it can be improved by using of a discrete solution space instead of the continuous one. What is more, the discrete set can be reduced to few paths that cover only a part of area around start point, in which physical constraints allows the robot to move. The number of the prototype paths affects quality and performance of the algorithm. More prototype paths increase quality and unfortunately increase the computation time, while fewer do exactly the opposite. The prototype paths number has to be adjusted to the computation platform performance and to the quality requirements.

In the method proposed in this paper, the prototype path exploits the control points that lie on the circular layers (Figure 8) around the start point and are distributed equidistantly on the both sides of the initial tangent vector.

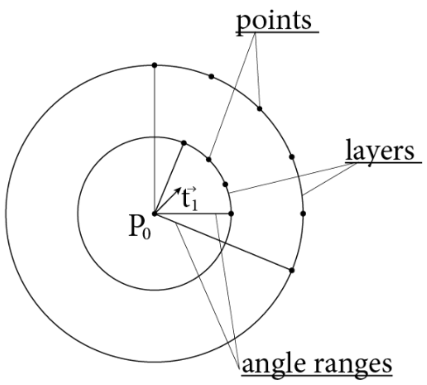

Figure 8: Layers description.

The number of layers is equal to number of control points that build the arcline. The first layer has only one point, which is the robot position. The initial tangent vector defines the orientation of the robot. Every next layer has to have equal or bigger amount of points. Points on every layer are distributed inside of the angle range (Figures 8, 9). The angle ranges can vary freely. They are adjusted experimentally, dependently on the environment in which the robot will operate.

$$\begin{array}{rccc} & L_0 & L_1 & L_2 \\ P: & [1 & 10 & 15] \\ A: & [\pi/2 & \pi/4 & \pi/2] \end{array}$$

Figure 9: Layers parameters: P – number of points, A – angle ranges, L – layers radii.

$$\begin{array}{l} L_0\text{: } [P_0] \\ L_1\text{: } [P_1\ P_2] \\ L_2\text{: } [P_3\ P_4\ P_5\ P_6\ P_7\ P_8] \\ \qquad\qquad \Downarrow \end{array}$$

$$\text{Paths family} \left\{ \begin{array}{l} [P_0\ P_1\ P_3] \\ [P_0\ P_1\ P_4] \\ [P_0\ P_1\ P_5] \\ [P_0\ P_2\ P_6] \\ [P_0\ P_2\ P_7] \\ [P_0\ P_2\ P_8] \end{array} \right.$$

Figure 10: Point layers to control points conversion.

After determination of positions of the points on the layers, they have to be gathered into control point lists that describe each path from the prototype paths set. Shorter layers are stretched to agree in length with the widest layer, as can be seen in Figure 10. The example of the set of the prototype paths is presented in Figure 11.

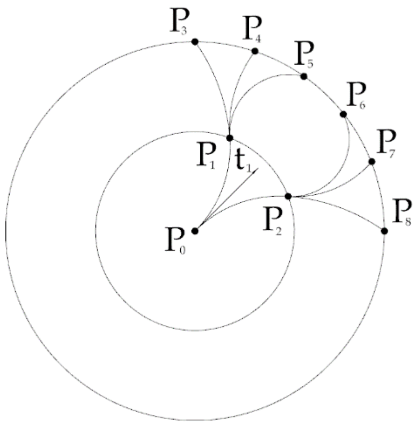

Figure 11: Example paths described by generated control points.

Finally, the best path is selected from the set (Figure10). It is done under two criteria: obstacle avoidance fitness and distance fitness. The distance fitness is the Cartesian distance between the global target point and the final point of the prototype path. It ensures that the path leads to the destination point. To obtain the obstacle avoidance fitness, the rasterization of the path is needed since it is described parametrically. A mechanism of security margin is applied by increasing path width by 2 pixels – 1 for each side of the path. Then the map status of corresponding pixels is checked. If any of path pixels collides on the map, the whole path is considered as incorrect path.

The algorithm returns the path from correct paths that has the lowest distance fitness value. It has to be noticed that this method is very computationally efficient and can be used as standalone algorithm for pathfinding in unknown environment.

6.3. Whole path generation algorithm – global path

The whole path generation algorithm builds the path using the primary path generation algorithm. The PPGA is used for setting the path in local area. Then the current robot position is moved along the local path by a given distance. The length of the local path has to be equal or longer than the distance of the movement. This process repeats until the distance to the target point is shorter or equal to the maximum radius of the PPGA radii layers. Then the layers radii are scaled down to make the maximum radius equal to the distance to the target point. Execution of the algorithm ends when the destination point is reached within given tolerance range. The points obtained at every step are sequentially added to the way point list. The WPGA is presented below in the form of list of steps.

1. Until the destination point is reached:
   1.1. Scale the radii of the layers of PPGA if the maximal radius is equal or lesser than distance between current position and target point to match to this distance.
   1.2. Execute the PPGA for current position.
   1.3. Set a point at given step distance on the obtained local path.
   1.4. Add the obtained point to the way point list.
   1.5. Set the obtained point as a new current point and the tangent vector at this point as a new robot orientation.
2. Return the result path as way point list.

The WPGA needs the following set of parameters to work:

- Start and destination points
- Start orientation
- Search radii list
- Search angle ranges list
- Point number on each layer list
- Obstacle list

6.4. Parameter set generation

The WPGA parameters such as start and destination points, start orientation and obstacle list are fundamental for every path definition and remain unmodified during the optimization process because they are environment conditions. The rest of them are used for the path shape definition. Although, they are not suitable for use in optimization methods like the GA, due to the fact, that they are lists and their length can vary. Moreover, more parameters mean wider solution space which affects the performance negatively. To overcome this inconvenience, the parameter list should be generated. It has to be based on as small as possible number of its parameters and be fast and easy to compute.

Thus, the number of layers is set as 3. The parameter lists are computed then by following formulae (13,14,15). Layers of radii S_{radii} are described by formula (13), layers of angle ranges S_{ranges} by (14) and layers of point numbers S_{points} by (15).

$$S_{radii} = \left[0, \frac{R_{max}}{2}, R_{max}\right] \tag{13}$$

$$S_{ranges} = \left[\frac{\pi}{4}, \frac{\delta_{max}}{2}, \delta_{max}\right] \tag{14}$$

$$S_{points} = [1, \mathrm{round}(0.8P_{max}), P_{max}] \tag{15}$$

As it can be seen, only three parameters are needed to generate these lists. They are maximum radius R_{max}, maximum angle range δ_{max} and maximum point number P_{max}.

7. Evaluation of path fitness

Evaluation of path fitness can be done in two ways. The first one is the generation of velocity profile and then use the obtained travel time as a fitness value. The second one is the estimation of quality of the path based on its shape. In experiments both methods are compared.

7.1 Estimation of path fitness

One of the possibilities of computation time reduction is the simplification of the most frequently used parts of the method. Optimization methods do not need the exact physical values. They need just to compare solutions and choose the best ones. Hence, the approximation of fitness can be used as well. The proposed estimation bases on three parts, which describes how fast can be given path (Eq.16). The first one is distance – shorter path is assumed to be faster than longer one. Thus, the quantity of distance of the path is equal to the sum of lengths of all path segments (Eq. 17). The second part is speed. As can be seen in Eq. (18), it depends on one path parameter only: radius. Bigger radius means higher maximum speed on the path segment. Therefore, the quantity of speed equals to sum of the radii of all path segments (Eq. 18). The last part that affects travel time is acceleration. The acceleration quantity is approximated by adding up differences between every pair radii of next segments (Eq. 19).

$$f(o) = d(o)c_d + s(o)c_s + a(o)c_a \tag{16}$$

$$d(o) = \sum_i A_l^{o_i} \tag{17}$$

$$s(o) = \sum_i A_r^{o_i} \tag{18}$$

$$a(o) = \sum_i A_r^{o_{i+1}} - A_r^{o_i} \tag{19}$$

In Eq. (16) c_d, c_s, c_a are coefficients of distance, speed and acceleration respectively. In Eqs (17, 18, 19) $A_l^{o_i}$ means the length of i-th segment of the o path and $A_r^{o_i}$ means the radius of i-th segment of the o path.

7.2. Velocity profile generation

Another method is generation of the velocity profile for the path. This method gives simulated travel time, not approximated fitness. The velocity profile generation algorithm is presented below as a list of steps.

1. Calculate maximum speed on arcs (Eq. 20).
2. Decrease the speed on slower parts (obtained from obstacle closeness detection, see section 7.3) by 50%.
3. Set maximum robot speed for arcs which maximum speed is higher than max robot speed.
4. Find the points where higher speed changes to lower speed (P_{HL}).
5. Initialize robot position on the path (distance 0 m) and speed (0 m/s). Initialize robot speeds list as an empty list and simulation time list as a list with first element equal to 0.
6. Repeat until the end of the path is not reached:
    - 6.1. Calculate deceleration distances ($d_{distance}$, see Eqs (21, 22, 23)) to every P_{HL} point for current speed.
    - 6.2. If any of the deceleration distances covers current robot position on the path and speed is higher than corresponding maximum speed of the arc of P_{HL} point then enable deceleration. Otherwise if current speed is lower than maximum speed on current path arc then enable deceleration.
    - 6.3. If after acceleration, speed will not exceed current maximum speed and acceleration is enabled then accelerate.
    - 6.4. If after deceleration, speed will not exceed 0 m/s and deceleration is enabled then decelerate.
    - 6.5. Update robot speed and position on the path. Update simulation time. Add robot speed to the end of the speeds list and add current simulation time to the simulation time list.

Maximum speed on arc is computed for each path segment. For the straight segments it is equal to maximum robot speed. For arc segments it is computed by Eq. 20. Maximum velocity for arcs is formulated from friction and centrifugal forces. It is a square root of friction coefficient μ, standard gravity g and arc radius A_r. The ground is assumed to be always flat.

$$v_{max} = \sqrt{\mu g A_r} \tag{20}$$

Deceleration distances are calculated using following formulae (21, 22, 23). The Δa is given acceleration, the Δt is given step size, the v_c is current step speed and the v_t is target speed.

$$a_{\Delta} = \Delta a \cdot \Delta t \tag{21}$$

$$t_{\Delta} = \frac{v_c - v_t}{a_{\Delta}} \cdot \Delta t \tag{22}$$

$$d_{distance} = \frac{\Delta a \cdot t_{\Delta}^2}{2} + v_c \cdot t_{\Delta} \tag{23}$$

The result travel time is the value of the last element of the simulation time list. The robot speeds list contains the velocity profile for the given path. In Figure 12 can be seen an example of a velocity profile, in which can be easily recognized parts of acceleration, constant speed and deceleration.

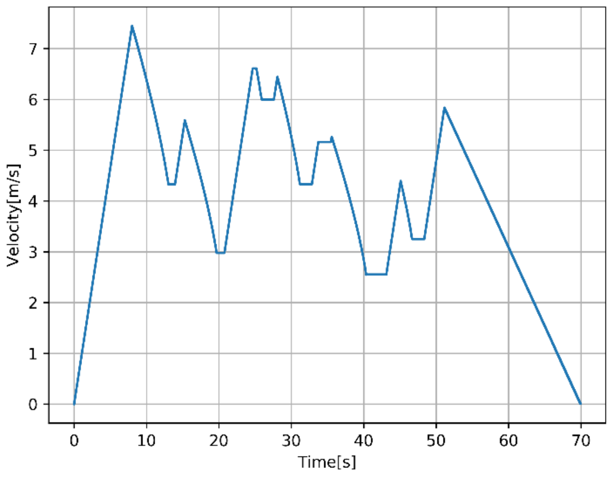

Figure 12: An example of a velocity profile obtained by proposed velocity profile generator.

7.3. Closeness ranges detection

Closeness ranges are ranges on the path length in which the obstacles lay close to the path. Basing on the heuristic knowledge gained by humanity by centuries, it is better to slow down, when the human or vehicle passes close to an obstacle. This behaviour provides more rigidity to errors in driving when the possibility of collision is higher. In the velocity profile generation this approach is used. The maximum velocity in closeness ranges is reduced by 50% to maximum speed implied from the path properties.

In the proposed method, the closeness ranges are detected by obstacles detection on the map in the window of given size (bigger than 1x1). The window is moved along rasterized path pixel by pixel and when an obstacle is detected in the window, the current point is saved as start point of the range. Then the window continues the movement and when it reaches the window with no obstacles, the position is saved as end point of the range. Finally, both positions are added together to the list of closeness ranges. The process continues until the end of the path is reached by the window. As a result, the list of closeness ranges is obtained.

8.Computer simulations

To compare efficiency of the described methods, the computer tests were carried out.

8.1. Methodology

For the purpose of the test, three different maps are used (Figure 13). The first map simulates the forest environment with the randomly distributed trees. The second simulates area with some

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0564.v1

buildings or boxes. The third map is designed to test the methods in some kind of complex environment, in which it is necessary to produce a complex path with many bends. The start point is marked by blue cross inside of the blue circle and the target point is marked by yellow cross inside of the yellow circle. In the maps with paths the start and target points markings are omitted.

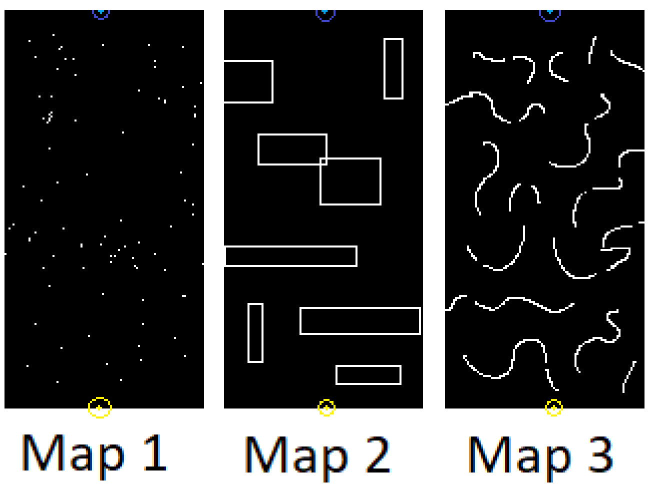

Figure 13: Maps used for the tests.

There are two methods: the genetic algorithm and the random-search, each in two versions: first using an estimated path fitness and second using travel time fitness from velocity profile generator. It gives four algorithms to be tested. For each test, two values are saved: time of algorithm execution and result travel time. The travel time is computed using mentioned velocity profile generator. All the tests were performed on the same platform: PC computer with AMD Ryzen 5 3600X processor. The algorithms are implemented in Python 3.6. All the computations are performed in single thread. The robot always starts in the middle of the top of the map and the target point is in the middle of the bottom of the map. Each map is of the same dimensions – 100x200 px.

The robot maximum speed was set to $v_{max} = 10\frac{m}{s}$ and the friction coefficient is set to 0.3. Other significant parameters of the methods are collected in the Tables 6 and 7.

8.2. Results

The results are collected in Table 1. Table 2 shows average times of execution and travel for each map. In Table 3 the standard deviations of execution and travel times are presented. In Tables 4 and 5, the percentage difference in times can be seen. The times of genetic algorithm-based method in fitness estimation variant are used as a reference. Following abbreviations are used in the tables: 'GA' – genetic algorithm, 'ex.' – execution, 'VP' – velocity profile, 'RS' – random search.

Figure 14 shows example paths on maps 1, 2 and 3 obtained by random search with velocity profile method. Figures 15, 16, and 17 present velocity profiles for those paths respectively.

Values in the table 4 and 5 are computed using formula (24), where d_f is difference in percent, t_c is time to be compared, t_{GA} is time of the GA method. Thie formula is used for both execution and travel times.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0564.v1

$$d_f = \frac{t_c - t_{GA}}{t_{GA}} \cdot 100\% \tag{24}$$

Table 1: Experiments results: times of execution and result times of the result paths.

Map	GA ex. time [s]	GA travel time [s]	GA VP ex. time [s]	GA VP travel time [s]	RS ex. time [s]	RS travel time [s]	RS VP ex. time [s]	RS VP travel time [s]
1	34,36	41,65	58,44	41,48	3,27	41,75	3,245	41,49
	29,043	41,55	80,17	41,48	1,874	42,01	2,606	41,53
	49,585	41,54	65,29	41,47	2,488	41,49	3,81	41,5
	33,271	41,62	78,62	41,55	2,42	41,76	2,878	41,49
	39,666	41,57	72,26	41,49	3,125	41,53	2,491	41,62
2	68,04	51,44	112,52	50,96	2,45	52,63	1,605	52,04
	87,458	51,41	88,77	49,03	3,548	56,26	2,202	47,02
	55,26	51,98	74,07	51,5	1,929	48,83	7,741	49,67
	31,611	54,83	151,57	50,84	2,988	54,62	2,691	56,15
	54,94	52,24	132,26	51,35	1,833	53,44	2,166	47,16
3	47,98	60,76	29,8	52,03	2,379	50,55	1,609	51,64
	25,499	46,48	120,48	51,82	1,678	43,47	3,55	49,68
	32,712	54,61	110,38	42,82	3,077	51,04	2,529	49,37
	61,075	51,96	82,96	50,73	2,653	54,03	2,063	53,4
	29,755	69,51	112,14	51,19	3,778	50,55	1,375	66,14

Table 2: Average times of algorithm execution and travel.

Map	GA ex. time [s]	GA travel time [s]	GA VP ex. time [s]	GA VP travel time [s]	RS ex. time [s]	RS travel time [s]	RS VP ex. time [s]	RS VP travel time [s]
1	37,185	41,586	70,956	41,494	2,6354	41,708	3,006	41,526
2	59,4618	52,38	111,838	50,736	2,5496	53,156	3,281	50,408
3	39,4042	56,664	91,152	49,718	2,713	49,928	2,2252	54,046

Table 3: Standard deviation of times of algorithm execution and travel.

Map	GA ex. time [s]	GA travel time [s]	GA VP ex. time [s]	GA VP travel time [s]	RS ex. time [s]	RS travel time [s]	RS VP ex. time [s]	RS VP travel time [s]
1	7,90	0,05	9,14	0,03	0,57	0,21	0,54	0,06
2	20,44	1,41	31,44	0,99	0,72	2,78	2,52	3,81
3	14,79	8,83	37,09	3,89	0,78	3,89	0,86	6,95

Table 4: Percentage difference between travel times of the methods. The GA times as a reference.

Method	Map 1	Map 2	Map 3
GA	0%	0%	0%

GA VP	-0.22%	-3.14%	-12.26%
RS	0.29%	1.48%	-11.89%
RS VP	-0.14%	-3.76%	-4.62%

Table 5: Percentage difference between execution times of the methods. The GA times as a reference.

Method	Map 1	Map 2	Map 3
GA	0%	0%	0%
GA VP	90.82%	88.08%	131.33%
RS	-92.91%	-95.71%	-93.11%
RS VP	-91.92%	-94.48%	-94.35%

Table 6: The parameters of the GA-based methods.

Parameter	Value
Population size	20
Step size	10
Epochs	10
Maximum radius	50
Maximum angle range	$\frac{\pi}{1.2}$
Maximum points number	50

Table 7: The parameters of RS-based methods.

Parameter	Value
Epoch number	5
Step size	10
Maximum radius	50
Maximum angle range	$\frac{\pi}{1.2}$
Maximum points number	50

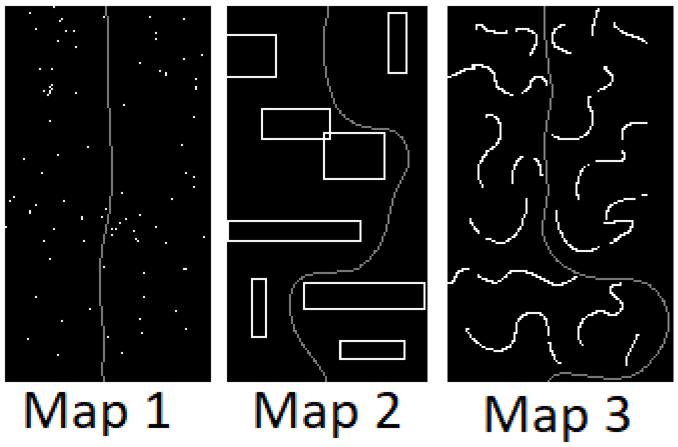

Figure 14: Examples of generated paths.

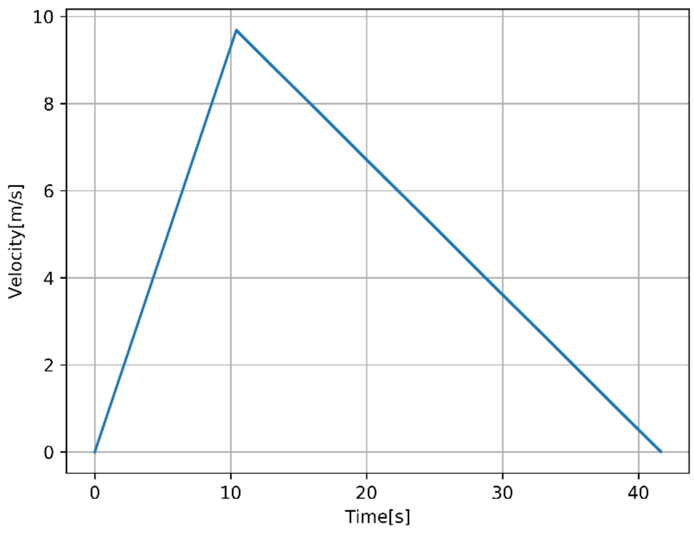

Figure 15: Velocity profile for the example path for the map 1.

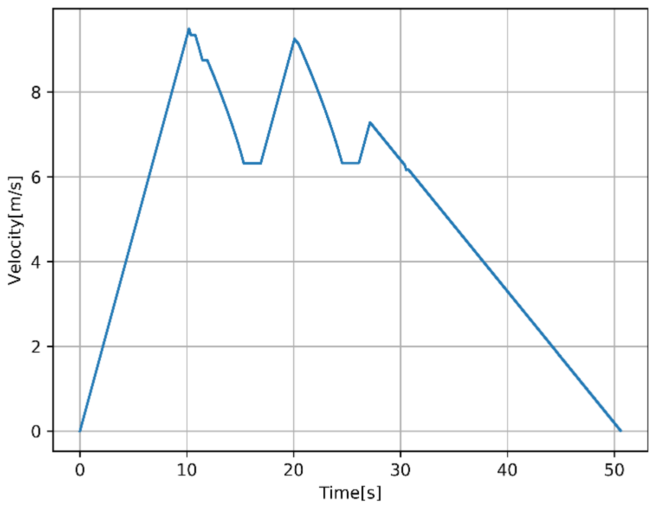

Figure 16: Velocity profile for the example path for the map 2.

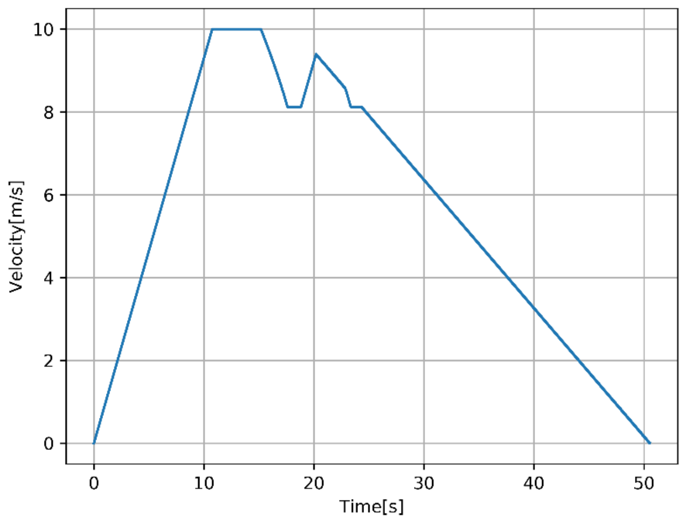

Figure 17: Velocity profile for the example path for the map 3.

The first observation is that random search methods are generally much more time efficient than genetic algorithm-based ones. As can be seen in Table 5, the execution times are better for more than 90%. In comparison to the GA algorithm with fitness estimation, for simple maps as map 1, the quality of the results of other methods do not differ much. The difference is more significant as the map complexity increases. The methods that use the velocity profile generator gives better results. Although the RS method with fitness estimation gives worse results for map 1 and 2, for map 3 it is better for over 10% and the result quality is almost the same as for the GA VP method. What is surprising, the RS VP result for the most complex map is worse than two mentioned methods and is better than the GA only for approx. 4%. Generally, it can be said, that the best method according to travel time is the GA VP. Unfortunately, it is also the method of the worst performance. For the most complex map, its execution time is worse for almost 130% than the GA, when the random search-based methods are better for over 90%. It is worth to notice that the RS based methods have more stable execution times (see Table 3) than the GA-based ones. Comparing that with absolute times of execution from Table 2, it can be seen that execution times of the RS methods are from range 2-3.5, when those of the GA are up to 50 times longer. In the case of the RS-based methods, the difference between execution times of the fitness estimation and travel time are not significant. In the GA-based ones however, average execution times differ even for over 100%.

9. Conclusion and future work

The random search-based methods provide much better performance than genetic algorithm-based ones. What is interesting, there is no big difference between using estimation of fitness and travel time from velocity profile generator. Difference is significant (over 10%) in case of GP VP, however similar difference has RS method, which uses estimation. Ergo the quality of optimization depends more on optimization method than on fitness obtaining method. It shows also that proposed way of estimation gives a proper knowledge of way properties. It is surprising that, in the most complex case, the estimated fitness (in RS) gave results close to the GP based on travel time, because estimated fitness does not take closeness ranges into consideration. Another interesting fact is a big gap between execution times of GA-based methods and RS-based ones. It will be investigated, if it depends on method character only or maybe there are implementation issues that slows it.

The next step of the research is to conduct experiments in real environment. For this purpose, an ATV-based mobile platform is under construction. It is powered by a combustion engine and will be able to operate in various environments, such as forest. The goal is to make it to move through a quite rough terrain as fast as possible.

References

1. Szczerba, R. J.; Galkowski, P.; Glicktein, I. S.; and Ternullo, N. Robust algorithm for real-time route planning. *IEEE Transactions on Aerospace and Electronic Systems*, **2000**, Vol. 36, no. 3, pp. 869-878, doi: 10.1109/7.869506.
2. Zhang, J.; Hu, C.; Chadha, R.G.; Singh, S. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *Journal of Field Robotics*. **2020**; pp.1–14. https://doi.org/10.1002/rob.21952.
3. Kwaśniewski, K. K.; Gosiewski, Z. Genetic Algorithm for Mobile Robot Route Planning with Obstacle Avoidance. *Acta Mechanica et Automatica*, **2018**, Vol.12, no.2, pp.151-159. doi: https://doi.org/10.2478/ama-2018-0024
4. Zhang, H.; Liu, Z. 3D path planning for micro air vehicles based on quantum-behaved particle swarm optimization algorithm. *Journal of Central South University*, **2013**, Vol.44, pp.58-62.
5. Duan, H.; Huang, L. Imperialist competitive algorithm optimized artificial neural networks for UCAV global path planning, *Neurocomputing*, **2014,** Vol. 125, pp. 166-171, ISSN 0925-2312. https://doi.org/10.1016/j.neucom.2012.09.039
6. Bozek, P.; Karavaev, Y.L;. Ardentov, A.A.; Yefremov, K. S. Neural network control of a wheeled mobile robot based on optimal trajectories. *International Journal of Advanced Robotic Systems*. **2020**. https://doi.org/10.1177%2F1729881420916077
7. Yi, Z.; Yanan Z.; Xiangde, L. Path Planning of Multiple Industrial Mobile Robots Based on Ant Colony Algorithm, Proc.*16th International Computer Conference on Wavelet Active Media Technology and Information Processing*, **2019**, Chengdu, China, pp. 406-409, doi: 10.1109/ICCWAMTIP47768.2019.9067693.
8. Zhang, W.; Gong, X.; Han, G.; Zhao, Y. An Improved Ant Colony Algorithm for Path Planning in One Scenic Area With Many Spots, *IEEE Access* **2017**, Vol. 5, pp. 13260-13269, , doi: 10.1109/ACCESS.2017.2723892.
9. Wang, T.; Dang, Q.; Pan, P. Path planning approach in unknown environment. *Int. J. Autom. Comput.* Vol. 7, pp.310–316, 2010. https://d oi.org/10.1007/s11633-010-0508-6
10. Xiang, C.; Fen, Z. A fuzzy-based potential field hierarchical reinforcement learning approach for target hunting by multi-AUV in 3-D underwater environments, *International Journal of Control*, **2019,** DOI: 10.1080/00207179.2019.1648875

11. Orozco-Rosas, U.; Picos, K.; Montiel, O. Hybrid Path Planning Algorithm Based on Membrane Pseudo-Bacterial Potential Field for Autonomous Mobile Robots, *IEEE Access*, **2019 ,**Vol. 7, pp. 156787-156803, doi: 10.1109/ACCESS.2019.2949835.
12. Roberge, V.; Tarbouchi, M.; Labonté, G. Fast Genetic Algorithm Path Planner for Fixed-Wing Military UAV Using GPU. *IEEE Transactions on Aerospace and Electronic Systems*, **2018**, Vol. 54, no. 5, pp. 2105-2117, Oct., doi: 10.1109/TAES.2018.2807558.
13. Liu, Y.; Guo, C.; Weng, Y. Online Time-Optimal Trajectory Planning for Robotic Manipulators Using Adaptive Elite Genetic Algorithm With Singularity Avoidance. *IEEE Access* **2019**, Vol. 7, pp. 146301-146308, doi: 10.1109/ACCESS.2019.2945824.
14. Jiang, A.; Yao, X.; Zhou, J. Research on path planning of real-time obstacle avoidance of mechanical arm based on genetic algorithm. *The Journal of Engineering*, **2018**, Vol. 11, no. 16, pp. 1579-1586, doi: 10.1049/joe.2018.8266.
15. Lo, C.; Yu, S. A two-phased evolutionary approach for intelligent task assignment & scheduling, *11th International Conference on Natural Computation (ICNC)*, Zhangjiajie, **2015**, pp. 1092-1097, doi: 10.1109/ICNC.2015.7378144.
16. Zhang, J.; Zhang ,Y.; Zhou, Y. Path Planning of Mobile Robot Based on Hybrid Multi-Objective Bare Bones Particle Swarm Optimization With Differential Evolution, *IEEE Access*, **2018**, Vol. 6, pp. 44542-44555, doi: 10.1109/ACCESS.2018.2864188.
17. Li, G.; Yamashita, A.; Asama, H.; Tamura, Y. An efficient improved artificial potential field based regression search method for robot path planning. *2012 IEEE International Conference on Mechatronics and Automation*, Chengdu, **2012**, pp. 1227-1232, doi: 10.1109/ICMA.2012.6283526.
18. Jahanshahi, H.; Jafarzadeh, M.; Sari, N.N.; Pham, V.-T.; Huynh, V.V.; Nguyen, X.Q. Robot Motion Planning in an Unknown Environment with Danger Space. *Electronics* **2019**, Vol.8, no.2, doi.org/10.3390/electronics8020201.
19. Lee, H.-Y.; Shin, H.; Chae, J. Path Planning for Mobile Agents Using a Genetic Algorithm with a Direction Guided Factor. *Electronics* **2018**, Vol.7,p. 212, https://doi.org/10.3390/electronics7100212.
20. Kwaśniewski, K. K.; and Gosiewski, Z. Wheeled Robot Path Planning in Natural Environment, *2020 International Conference Mechatronic Systems and Materials (MSM*), Bialystok, Poland, **2020**, pp. 1-6, doi: 10.1109/MSM49833.2020.9202394.