NON - EUCLIDEAN METRIC AND PATH PLANNING

Edvards Valbahs[1] and Peter Grabusts[2]

Rezekne Academy of Technologies, Rezekne, Latvia

ABSTRACT

In order to achieve the wide range of the robotic application it is necessary to provide iterative motions among points of the goals. For instance, in the industry mobile robots can replace any components between a storehouse and an assembly department. Ammunition replacement is widely used in military services. Working place is possible in ports, airports, waste site and etc. Mobile agents can be used for monitoring if it is necessary to observe control points in the secret place. The paper deals with path planning programme for mobile robots. The aim of the research paper is to analyse motion-planning algorithms that contain the design of modelling programme. The programme is needed as environment modelling to obtain the simulation data. The simulation data give the possibility to conduct the wide analyses for selected algorithm. Analysis means the simulation data interpretation and comparison with other data obtained using the motion-planning. The results of the careful analysis were considered for optimal path planning algorithms. The experimental evidence was proposed to demonstrate the effectiveness of the algorithm for steady covered space. The results described in this work can be extended in a number of directions, and applied to other algorithms.

KEYWORDS

Path Planning, Robotic, Simulated Annealing.

1. INTRODUCTION

The article is connected to the travelling salesman problem (TSP), but with some exceptions and conditions. In the case when the TSP is envisaged the following approximate path planning algorithms are used [1, 2, 3]:

- The closest neighbour algorithm;
- Simulated Annealing (SA);
- Genetic Algorithm (GA);
- Ant colony optimization.

The closest neighbour approach is the simplest and straightforward TSP one [4]. The way to this approach to always visit the closest city. The polynomial complexity of the approach is $O(n^2)$. The algorithm is relatively simple:

- choose a random city;
- find out the nearest city unvisited and visit it;
- are there any unvisited cities left? If yes, repeat step 2;
- return to the first city.

SA is successfully used and adapted to get an approximate solutions for the TSP [4]. SA is basically a randomized local search algorithm similar to Tabu Search but do not allow path exchange that deteriorates the solution. The polynomial complexity of the approach is $O(n^2)$ accordingly.

Input: ProblemSize, $iterations_{max}$, $temp_{max}$
Output: S_{best}
1. $S_{current} \leftarrow$ CreateInitialSolution(ProblemSize)
2. $S_{best} \leftarrow S_{current}$
3. **for** $i=1$ **to** $iterations_{max}$ **do**
4. $S_i \leftarrow$ CreateNeighborSolution($S_{current}$)
5. $temp_{curr} \leftarrow$ CalculateTemperature(i, $temp_{max}$)
6. **if** Cost(S_i)≤Cost($S_{current}$) **then**
7. $S_{current} \leftarrow S_i$
8. **if** Cost(S_i) ≤ Cost(S_{best}) **then**
9. $S_{best} \leftarrow S_i$
10. **end**
11. **else if** Exp((Cost($S_{current}$)-Cost(S_i))/$temp_{curr}$)>Rand() **then**
12. $S_{current} \leftarrow S_i$
13. **end**
14. **end**
15. **return** S_{best}

Figure 1. Pseudocode for SA

The SA method [5, 6] is widely used in applied science (see Figure 1). The well-known traveling salesman problem has effectively solved by means of this method. For instance, the arrangement of many circuit elements on a silicon substrate is considerably improved to reduce interference among the elements [7, 8].

GA conducts in a way similar to the nature [2]. A basic GA starts working with a randomly generated population of potential solution. The candidates are then mated to produce offspring and only some of them go through a mutating process. Each candidate has an optimal value demonstrating us how go it is. Choosing the most optimal candidates for mating and mutation the overall optimality of the candidate solutions increases. Using GA to the TSP involves implementing a crossover procedure, a measure of optimality and mutation as well. Optimality of the solution is a length of the solution.

Ant colony optimization is the algorithm that is inspired by the nature [9]. It is based on ant colony moving behaviour. Good results can be achieved by means of the algorithm but not for complex problems.

We managed to use SA algorithm rather successfully in our previous work [10] taking into account the specific side of this work (it will be discussed in detail further). Therefore, it is necessary to discuss some principles of SA realization in detail. In order to calculate the total path it is necessary to know the shortest route among all the cities. As we do not know the distance, we must apply one of the algorithms to define the shortest route among all the cities. It is Dijkstra's algorithm [11] that gives the possibility to get the shortest path tree. The polynomial complexity of the Dijkstra's algorithm is $O(n^2)$.

2. GOALS

The aim of the research paper is to analyze motion-planning algorithms that contain the design of modelling programme. The programme is needed as environment modelling to obtain the simulation data. The simulation data give the possibility to conduct the wide analyses for selected algorithm. Analysis means the simulation data interpretation and comparison with other data obtained using the motion-planning.

The use in practice and the necessity of it is greatly connected to optimal algorithm and methodological work out for autonomous agents that move in the space and are able to plan route

on their own [12, 13, 14, 15, 16, 17]. One of such agent-samples exiting in our everyday life is autonomous vacuum cleaner. Autonomous vacuum cleaners do not usually use the motion-planning algorithm. They are based on some simple algorithms, for example cleaning in a spiral, crossing the premises avoiding the walls and their moving is casual after touching the walls. The philosophy of this design was offered by the scientists of Massachusetts Institute of Technology. Agents must behave as insects having primitive controlling devices in accordance to the environment. As a result, though an autonomous vacuum cleaner is very effective in cleaning premises, it is required much more time as compared with work made by a human. There is a drawback, the autonomous vacuum cleans some spaces many times but other spaces only once. The use of motion-planning algorithms can raise the effectiveness of an autonomous vacuum cleaner.

3. ASSUMPTIONS

In order to fulfill the aim of the research paper the following conditions are introduced for:

- premises where an object moves;
- robot (or object) moves around the premises;
- path the robot moves on in the premises.

The premises are presented as two-dimensional plane. The plane of premises is equally divided into the cells. The cell dimensions are equal to agent dimension that moves in the premises. The space can be represented as a graph with two kinds of edges (see Figure 2). Horizontal and vertical edges are marked with unbroken lines they are of similar length, but others are longer and marked with dash lines. It is linked with agent movement possibilities.

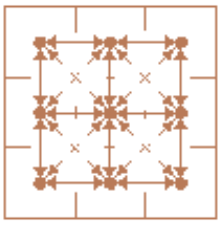

Figure 2. The example of the graph and 3 x 3 space

The object moves only one cell forward and back i.e. during one motion the object can move to the one cell from empty eight ones (eight cells around one cell) paying attention to that cell is not occupied by the obstacle but if it is occupied, the graph will not have the relevant vertex (see Figure 3).

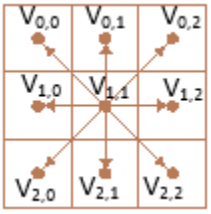

Figure 3. The example of agent's motion (where $v_{i,j}$ is relevant vertex)

As opposed to classical TSP we take a number of additional rules and it means that the agent can cross the one the same cell several times in succession (it must cross any cell one time obligatory). Thus, the agent's initial vertex does not coincide with its final vertex of total route.

In this research paper both algorithms were compared practically using and combining different placement of obstacles in the unchangeable two-dimensional space. All the results were obtained on one and the same computer (2.66 GHz processor and 2GB RAM), operating systems (Ubuntu 15.04 Linux were used). The following information was collected about:

- the number of covering for each cell;
- the time which was necessary for both algorithms to plan the route.

The given illustrations (see Figure 5) show coverage density (it is an example that was obtained in our previous work [10]).

Figure 4. Density scale (white - uncovered; black - covered the most)

The density scale (see Figure 4) is the same for all coverage densities. Coverage density shows how often the robot covers each cell.

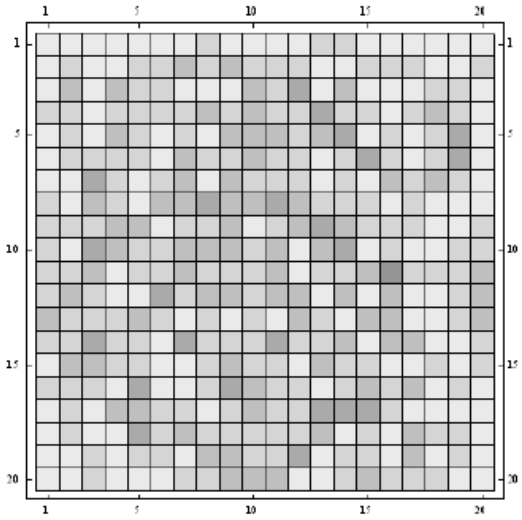

Figure 5. Coverage density for the space without obstacles for SA

4. MATHEMATICAL ANALYSIS

Let's discuss Euclidian (or hypotenuse of right-angled triangle) and non-Euclidian (calculated with offered/developed approach) distance between two graph points that are mutually moved along vertical and longitude (see Figure 6).

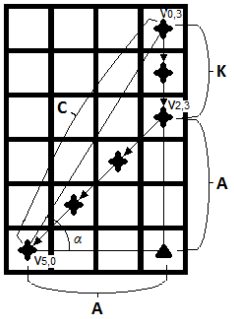

Figure 6. Example for agents moving from junction $v_{0,3}$ to junction $v_{5,0}$

Let's label the distance calculated by means of non-Euclidian geometry with *L*, then $L = K + 2^{0.5}*A$. You can see that these distances numerically differ and not always are equal. Usually *L* is a bit bigger than Euclidian distance (or hypotenuse *C*). *L* matches with Euclidian distance at following conditions (see Figure 6):

- if catheti of a triangle are equal and then K = 0 (in case of equilateral triangle);
- or junctions are not mutually moved along vertical or longitude. For example if one of the catheti of triangle is equal with zero.

To assess numerically the difference between both distances, you can invent the following coherence: *L* corresponds to 100% and we have to calculate for how many percent *L* is bigger than *C*, i.e., how big percent refers to *(L – C)* difference (let's label the difference percent from *L* with *P*, then *P = ((L-C)*100)/L*). Differences and *L* values depend on *K* and *A* values. Then *P* dependence on *K* and *A* can be expressed with a function with two arguments (see Figure 7).

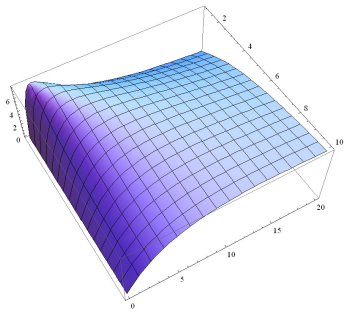

Figure 7. Example for agents moving from junction $v_{0,3}$ to junction $v_{5,0}$

If *A = const*, then *P* is a function with one argument that depends on *K* (see Figure 8).

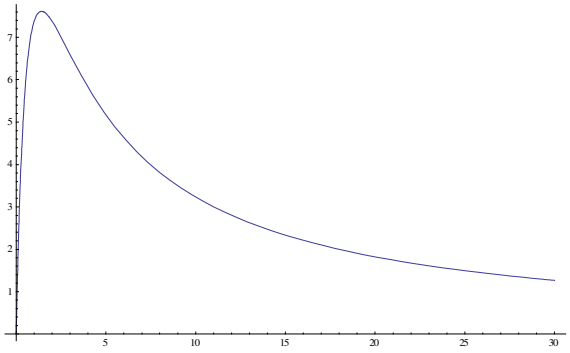

Figure 8. P dependence on K, if A = 1

You can see that at all *A* values maximum *P* value is equal with ~7.612%. You can express (analytically) *K* value from the following equitation (considering that $L = K + 2^{0.5}*A$): $(((K + 2^{0.5}*A)-C)*100)/(K + 2^{0.5}*A) = 7.611$. Solution of equitation is quite simple: $K = 1.414*A$ (it must be marked that *K* is a whole number, i.e., it will always be bigger or smaller that the given value). You can also calculate value of angle α at the maximum value *P* that will remain constant irrespective of value *A*. Considering that *K* is equal with $1.414*A$, then $\alpha = arcsin((A + 1.414*A)/(((A + 1.414*A)^2) + A^2)^{0.5}) = arcsin(2.414*A/(5.827*A^2+A^2)^{0.5}) = arcsin(2.414*A/(6.827*A^2)^{0.5}) = arcsin(2.414*A/2.613*A) = arcsin(2.414/2.613)$. After all simplifications angle α is equal with 67.65^O.

5. RESULTS

Taking into account the fact that the distance among all the vertexes (cities) are unknown in the beginning, it is necessary to define the shortest paths among those vertexes mentioned above. Dijkstra's algorithm can be used but increasing the measures of the premises, the time is proportionally increases accordingly that is necessary for evaluating path tree. Therefore, it is needed to simplify the calculation of the shortest path, which is possible, provided the peculiarities and nuances of the task are taken into consideration. In addition, the empty premises should be observed. If all the mentioned above remains valid, the simple algorithm can be worked out to define the shortest paths.

Let us consider the agent's general moving paths. If there are no vertexes between the current initial and goal vertexes, the agent can move only to eight possible positions (cells) depending on goal vertex (see Figure 3). Admitting that first vertex index i define the vertical position and the second vertex j defines the horizontal position we can draw a line either horizontally or vertically. And one of the vertexes will have the index with common value (see Figure 9).

Figure 9. The example of agent moving horizontally (where i index value is common for both vertexes)

Another situation can be seen if the current initial and goal vertexes are neither on the horizontal nor vertical lines (see Figures 10-12).

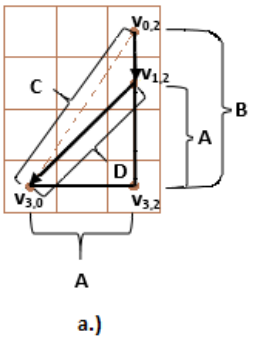

Figure 10. Three examples of agent moving (where A, B and C are sections among the vertexes): agent moves from $v_{0,2}$ to $v_{3,0}$ crossing $v_{1,2}$

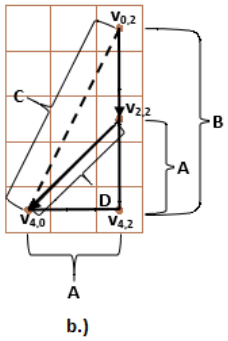

Figure 11. Three examples of agent moving (where A, B and C are sections among the vertexes): agent moves from $v_{0,2}$ to $v_{4,0}$ crossing $v_{2,2}$

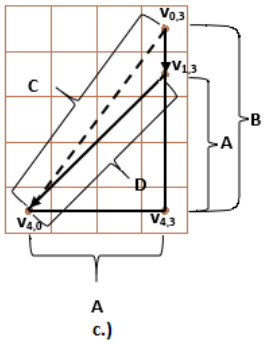

Figure 12. Three examples of agent moving (where A, B and C are sections among the vertexes agent moves from $v_{0,3}$ to $v_{4,0}$ crossing $v_{1,3}$)

All cases of Figures 10-12 have a common characteristic that unites them. The shortest path from initial vertex to goal vertex is section C but for the agent this path is unavailable because of current task conditions and peculiarities. These cases can be described by the right-angled triangle where C is a side of the triangle. In addition, side B is longer than side A. One of the shortest paths among the relevant (corresponding) vertexes:

- the agent moves along the longest side B of the right-angled triangle until the gap between the covered path and side B is equal to side A;

- if gap between the covered path and side B is equal to side A, then the agent moves along the angle allowed (along the section D) to the goal vertex (let us mark that this action corresponds to the case when side B is equal to side A i.e. the right-angled triangle is the isosceles triangle, too (see Figure 10) in case initial vertex is $v_{1,2,4}$, (see Figure 8) in case initial vertex is $v_{2,2}$ and (see Figure 12) in case initial vertex is $v_{1,3}$)).

We can follow that the path is longer than optimal side C. And it can be calculated by the use of following formulae: $L = B-A+2^{0.5}*A$, where L is the length of the path from initial vertex to goal vertex. By turn, C can be calculated from $C = (A^2+B^2)^{0.5}$. It is possible to calculate how match percent L is longer than C (if L is equal to 100 %), then the final result is equal to P=((L-C)*100)/L. Our goal premises are 100 x 100 cells. The value of P is reflected with contour line for the given premises depending on A and B (see Figure 13).

The method/algorithm mentioned was applied instead of Dijkstra's algorithm to calculate total path or covering of 100 x 100 cells big premises and it is obstacles free (see Figure 14).

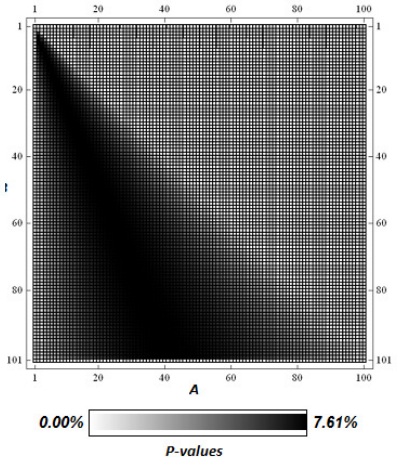

Figure 13. P value depending on B and A, if A > 1 and B > A

.

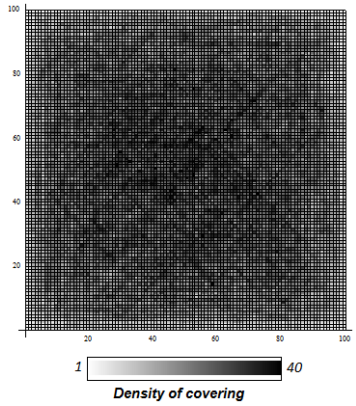

Figure 14. The density of covering for 100 x 100 cells big

Density of covering changes from 1 up to 40 (there are the cells which were covered only once and there are the cells covered maximum 40 times). Totally, the agent performed 192666 steps in order to cross each cell of the premises.

6. COMPARISON WITH OTHER SOLUTIONS

This section presents several search algorithms. Most of these are just classical graph search algorithms [18, 19].

Breadth first – selects states using the first-come, first-serve principle. This causes the search frontier to grow uniformly and is therefore referred to as breadth-first search. The asymptotic running time of breadth-first search is O(|V|+|E|), in which |V| and |E| are the numbers of vertices and edges, respectively, in the state transition graph [19].

Depth first – the resulting variant is called depth-first search because the search dives quickly into the graph. The running time of depth first search is also O(|V|+|E|) [19].

Dijkstra's algorithm – systematic search algorithm that finds optimal plans because it is also useful for finding feasible plans. The running time is O(|V| lg |V|+|E|), in which |V| and |E| are the numbers of edges and vertices, respectively, in the graph representation of the discrete planning problem [19].

A-star algorithm – works in exactly the same way as Dijkstra's algorithm. The only difference is the function used to sort a priority queue. It has some advantages, but in some problems it is difficult or impossible to find a heuristic that is both efficient to evaluate and provides good search guidance [19, 20, 21].

Best first – the priority queue is sorted according to an estimate of the optimal cost-to-go. The solutions obtained in this way are not necessarily optimal; therefore, it does not matter whether the estimate exceeds the true optimal cost-to-go, which was important to maintain optimality for A^* search. The worst-case performance of best-first search is worse than that of A^* search and dynamic programming [19].

Iterative deepening – approach is usually preferable if the search tree has a large branching factor. This could occur if there are many actions per state and only a few states are revisited. The iterative deepening method has better worst-case performance than breadth- first search for many problems [19, 21].

Our approach – implementation is very simple according to the task (which is envisaged in the chapter "Results" in detail). The running time is O(C), in which C is a constant. The value of running time is bounded by a value that does not depend on the size of the input.

7. CONCLUSIONS

It can be concluded that the algorithm offered is rather simple and it replaced Dijkstra's algorithm effectively according to the task. The algorithm allows decreasing the time of calculation, which is necessary to define the shortest route among graph vertexes.

The shortest path can be defined in a simple way (even in such cases mentioned in Figures 7-9), provided that it is necessary to know the gap between the indexes of initial and goal vertexes. For instance, if initial vertex is $v_{i1,j1}$ and goal vertex is $v_{i2,j2}$, the first gap is $\Delta_1=|i_1-i_2|$ and the second gap is $\Delta_2=|j_1-j_2|$. As to the next step, it is needed to calculate the biggest gap between both the gaps. The shortest path is equal to the biggest gap. For instance, Figure 7 reflects the shortest path which occupies 4 cells, but in other cases (see Figures 8-9) it is 5 cells big.

It must be marked that total path can be a bit longer it is connected to the specific task which was envisaged in the chapters "Assumptions" and "Results" in detail. The worst case can be evaluated theoretically for the premises of 100 x 100 cells. If we take into consideration that the total route will consist of path pieces, which are longer than 7.61 % in comparison with C value (see Figure 10), the total path will be longer than optimal 7.61 % (actually, it is the worst maximal variant. We must pay attention to the fact that SA provides only approximate solution).

The algorithm can be successfully used e.g. in autonomous public transport restricted by means of rules, technical requirements in autonomous robots and military equipment. In addition, the algorithm can be used in various computer games where a path planning is done in dynamic environment.

It is possible to conclude that the algorithm offered can be used in the different application areas not only for path planning of a robot.

REFERENCES

[1] Applegate, D. L., Bixby, R. E., Chvátal, V. & Cook, W. J., (2007) *The Traveling Salesman Problem*, Princeton University Press, Princeton, USA.

[2] Cook, W. J. (2011) *In Pursuit of the Traveling Salesman*, Princeton University Press, Princeton, USA.

[3] Davendra, D. (2010) *Traveling Salesman Problem, Theory and Applications*, Tech, Rijeka, Croatia.

[4] Johnson, D. S. & McGeoch, L. A. (1995) *The Traveling Salesman Problem: A Case Study in Local Optimization*.

[5] Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. (1983) "Optimization by Simulated Annealing", *Science*, 220, pp. 671-680.

[6] Otten, R. H. J. M. & Ginneken, L. P. P. P. (1989) *The Annealing Algorithm*, Kluwer Academic Publishers.

[7] Kirkpatrick, S. (1984) "Optimization by Simulated Annealing: Quantitative Studies", *Journal of Statistical Physics*, 34, pp. 975-986.

[8] Vecchi, M. P. & Kirkpatrick, S. (1983) "Global Wiring by Simulated Annealing", *IEEE Transaction on Computer Aided Design*," CAD-2, pp. 215-222.

[9] Dorigo, M. & Gambardella, L. M. (1996) *Ant Colonies for the Traveling Salesman Problem*, University Libre de Bruxelles, Belgium.

[10] Valbahs, E. & Grabusts, P. (2012) *"Motion Planning of an Autonomous Robot in Closed Space with Obstacles", Scientific Journal of RTU. 5. series., Datorzinatne. - 15. vol., pp. 52-57.*

[11] Dijkstra, E. W. (1959) "A note on two problems in connexion with graphs", *Numerische Mathematik*, v. 1, p. 269-271.

[12] Ashkenazi, V., Park, D. & Dumville, M. (2000) "Robot Positioning and the Global Navigation Satellite System", *Industrial Robots: An International Journal*, 27(6), pp. 419-426.

[13] Batavia, P. H. & Nourbakhsh, I. (2000) "Path planning for the Cye personal robot", *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS).

[14] Biswas, R., Limketaki, B., Sanner, S. & Thrun, S. (2002) "Towards Object Mapping in Dynamic Environments with Mobile Robots", *Proceedings of the Conference on Intelligent Robots and Systems* (IROS), Lausanne, Switzerland.

[15] Buhmann, J., Burgard, W., Cremers, A.B., Fox, D., Hofmann, T., Schneider, F., Strikos, J., & Thrun, S., (1995) "The Mobile Robot Rhino", *AI Magazine*, 16(1).

[16] Choset, H. (2000) "Coverage of Known Spaces: The Boustrophedon Cellular Decomposition", *Autonomous Robots*, 9:247-253, Kluwer.

[17] Siegwart, R., Nourbakhsh, I. R. & Scaramuzza, D. (2011) *Introduction to Autonomous Mobile Robots*, A Bradford Book The MIT Press Cambridge, Massachusetts London, England.

[18] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001) *Introduction to Algorithms*, MIT Press, Cambridge, MA.

[19] LaValle, S. M. (2006) *Planning algorithms*, Cambridge University Press, Cambridge, UK.

[20] Fikes, R. E. & Nilsson, N. J. (1971) "STRIPS: A new approach to the application of theorem proving", *Artificial Intelligence Journal*, 2, pp. 189–208.

[21] Pearl, J. (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Boston, MA.

AUTHORS

Edvards Valbahs was born in Riga, Latvia. He received his Mg.sc.ing. degree in Information Technology from Riga Technical University in 2011. Since 2009 he has been working at A/S "Exigen Services Latvia" as a Systems Analyst. He is a PhD student at Rezekne Academy of Technologies. His research interests include neural networks and path planning algorithms. His current research focuses on techniques for path planning.

Peter Grabusts was born in Rezekne, Latvia. He received his Dr.sc.ing. degree in Information Technology from Riga Technical University in 2006. Since 2014 he is a Professor at the Department of Computer Science, IEEE member. His research interests include data mining technologies, neural networks and clustering methods. His current research focuses on techniques for clustering and fuzzy clustering