MAZE EXPLORATION ALGORITHM FOR SMALL MOBILE PLATFORMS

ŁUKASZ BIENIAS KRZYSZTOF SZCZEPAŃSKI PIOTR DUCH

Institute of Applied Computer Science,
lukaszbienias@gmail.com

Abstract. An algorithm that enables efficient maze exploration is presented in the paper. The algorithm involves two phases: first the whole maze is explored in an ordered way and then, the shortest possible way out is determined. The algorithm has been derived in a way that combines main advantages of the two known labirynth-exploration algorithms: "Wall follower" and "Trémaux's algorithm". The algorithm has been tested using an autonomous vehicle, controlled by Arduino UNO, with two DC engines, ultrasonic sensors and gyroscope. It has been shown that the proposed approach provides a few crucial advantages with respect to already known solutions.

Key words. maze exploration, small mobile platforms, arduino

1 Introduction

An amount of standalone devices which operates fully autonomously is growing every day. Nowadays, technology development is driven by a need to delegate different tasks to robots instead of people. There are many reasons for that, from time saving, through increasing precision, to avoiding threats to life or health. Tasks where autonomous vehicles can become especially attractive is exploration of different hazardous areas, such as contaminated zones, collapsed buildings, mines [9], inaccessible enivronments such as sea bed [3] or volcanos [2] etc. Functionalities that need to be supported by autonomous vehicle computers, necessary for enabling unmanned exploration, include avoiding obstacles and path-finding.

One of the tasks that autonomous vehicles could be confronted with is mapping a maze and determining ways out of it. To research various maze exploration strategies, the authors developed a small laboratory setup, comprising different mazes and a mobile platform operating under control of various maze-exploration algorithms.

An objective of the paper is to present a novel algorithm for real-time maze mapping and shortest-path determination, which has been developed as an alternative to existing approaches. The algorithm has been implemented on the two-wheeled platform, controlled by Arduino Uno and equipped with a set of sensors, including encoders, ultrasound sensors and gyroscopes. Performance of the proposed method has been evaluated and compared with performance of alternative solutions.

A structure of the paper is the following. Selected existing maze solving algorithms are presented in Section 2. The proposed algorithm for labirynth exploration is described in Section 3 and the mobile platform used throughout experiments is presented in Section 4. Section 5 presents results of experimental evaluation of the considered algorithms and Section 6 summarizes the paper.

2 Maze solving algorithms

Currently two different types of maze solving algorithms can be distinguished. The first type of algorithms are designed to be used inside a maze. It means that the traveller, who is going to explore a maze, has no prior knowledge about its configuration. On the other hand, the second group of algorithms is designed to be used by a computer or a person, who knows maze structure and can analyze it off-line. In the latter case, the way out can be determined in the most efficient way, before entering the maze. All dead-ends, which actually pose the greatest challenge for algorithms of the first group, are automatically avoided.

Unfortunately, the real life conditions are not as comfortable as assumptions of the second type of algorithms. In the majority of cases, a traveller does not have any perspective on the whole area, which should be explored. As a consequence, algorithms useful in real life scenarios, fall into the first of the aforementioned categories.

2.1 Left Wall Follower

One of the most popular algorithms, used to find the way out of the maze is "Wall Follower"[7]. The algorithm has two variants "Left Wall Follower" ("LWF") and "Right Wall Follower" ("RWF"). The principle of operation is analogous for both methods, so in what follows the logic of "Left Wall Follower" is presented. The main idea of this algorithm is to continuously follow the left wall inside the maze until the way out is found. The corresponding algorithm involves the following steps:

Step 1: Sense the left wall.

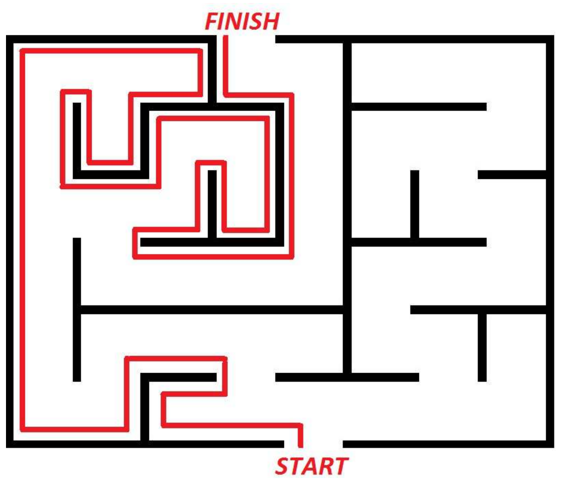


Fig. 1: "Left Wall Follower" solvable maze [7]

Step 2: If left wall present, then set *flag1* to 1, else set *flag1* to 0.

Step 3: If *flag1* is 1, then go to **Step 4**, else turn left by 90 degrees.

Step 4: Sense the front wall.

Step 5: If front wall present, then set *flagf* to 1, else set *flagf* to 0.

Step 6: If *flagf* is 0, then move straight, else turn right by 90 degrees.

Step 7: Return to **Step 1**.

Even though this logic ensures that the maze is traversed in an ordered way, it does not always guarantee that the way out will be found. If the exit from a labyrinth is placed inside the maze, while a structure of the maze contains free standing walls, a traveller can fall into an endless loop, which in turn would prevent from finding the solution. Sample result of applying "LWF" algorithm on a simple maze is presented in Fig. 1.

2.2 Pledge Algorithm

"Pledge Algorithm" is a modified version of the previously described "Left Wall Follower" algorithm [6]. The main advantage of this method comparing to the "LWF" is the fact that "Pledge Algorithm" is able to find way out of any labyrinth, even if the maze is disjointed and the solver starts inside the maze. Falling into an infinite loop is impossible, thanks to the ability of jumping between "islands". As a consequence, "Pledge Algorithm" can find its implementation for maze escaping robots.

The first step of the method it to pick a direction and always move towards it when possible. As soon as wall is faced, the "Left Wall Follower" algorithm is used until the chosen direction is available again. As long as "Left Wall Follower" method is used, the angles turned are counted. When the solver is facing the original direction again, and the angular sum of the turns made is 0, the solver leaves the obstacle and continues moving in its original direction. The counting ensures the traveller to be able to reach the far side of the island he is currently on, and jump to the next island in the chosen direction. This algorithm will keep on island jumping in that direction until the boundary wall will be faced, at which point "Left Wall Follower" takes the traveller to the exit.

This algorithm allows a user to find his way from any point inside to an outer exit of any finite two-dimensional maze, regardless of the initial position of the solver. However, this algorithm will not work in doing the reverse, which would be finding the way from an entrance on the outside of a maze to some end goal within it.

2.3 Flood Fill Algorithm

Another very interesting approach in solving mazes is provided by "Flood Fill Algorithm". To best understand the principle of operation of this algorithm, it can be imagined that the solver stands in the maze, which has non-permeable walls and flat level floors. The hosepipe is situated in the entrance of the maze and water starts to fill the maze from this point. The shortest path to the exit will be indicated by the first drop of water that arrives there. Fig. 2 presents an example of maze solved by this method.

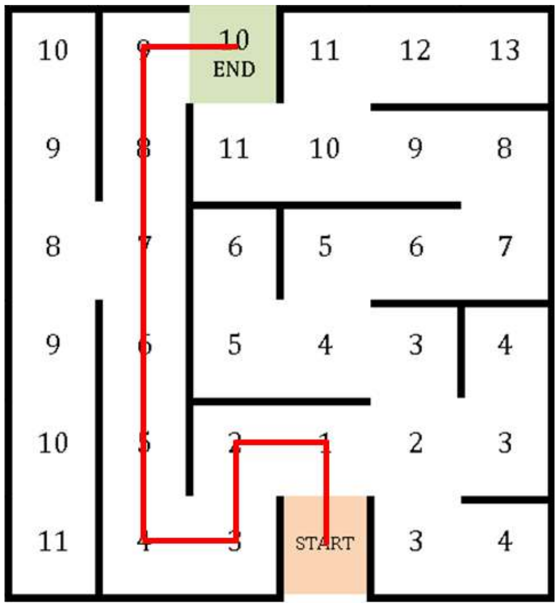


Fig. 2: "Flood Fill Algorithm" sample maze solved

Solution is found by starting at the destination cell and following the path of decreasing numbers. At this point, it should be clear why the algorithm is called flood fill. This method not only finds all possible solutions, but also indicates the best one and ensures exploration of the whole maze [7]. Although this algorithm seems very effective, its implementation on the mobile platform is difficult, as it requires knowledge of the structure of the maze before entering into it.

2.4 Tremaux's Algorithm

As it was stated before, "Left Wall Follower" strategy enables finding the way out, in general it is beneficial also to learn explored environments, as this would provide efficient behavior in potential future tasks related to the maze. Mobile platform, controlled by "LWF", would not be able to draw conclusions from its previous experiences.

Several more sophisticated algorithms have been introduced so far. One of them is "Trémaux's algorithm", which was proposed by Charles Pierre Tremaux in the nineteenth century [8]. In its original version, the algorithm requires marking an already travelled path with a line on a floor [7]. The most important rule of this approach is that no path can be traversed more than twice. When a traveller finds a dead-end or the junction, which is already marked, he has to turn around and go back the way he came. The last assumption prevents from moving around in circles. Subsequently, when a new junction is found, direction in which he has to follow is chosen randomly. Three types of passages can be distinguished in Trémaux's algorithm:

- empty, which means that the passage wasn't explored yet,
- marked once, pointing that the path was visited exactly once,
- marked twice, meaning that traveller was going that way and was forced to turn back, which indicates the wrong path.

Two different results of the maze exploration are possible. The first one is when a solution is found - the path marked exactly once indicates a way out of the maze. The second result describes the case, in which there is no exit from the maze. Such situation occurs when at the same time the traveller comes back to the start point and all paths are marked twice. Fig. 3 presents exploration history using "Trémaux's algorithm" in the exemplary maze.

There are many advantages of "Trémaux's algorithm", which makes it worth consideration for real life situations. The method is designed to be used by a traveller who is inside the maze. Moreover, a logic of the algorithm guarantees to work for any maze. Another benefit of this logic is the fact that the way out of a maze is mapped, in other words, "remembered". During the second and subsequent visit in a maze, it will be possible for a traveller to head directly to the exit of the maze, following the saved way and avoiding all dead-ends.

On the other hand, a few disadvantages can be pointed out. Firstly, the logic assumes random choice of direction, which should be followed in case of more than one free passage going out of one junction. This introduces some disorder, which can be avoided. Secondly, the algorithm requires painting a line to mark the path, which in real life conditions might be a problem. Thirdly, as it was already mentioned, there is a possibility that not whole maze will be traversed, which in turn may result in missing the shortest paths to the same exit.

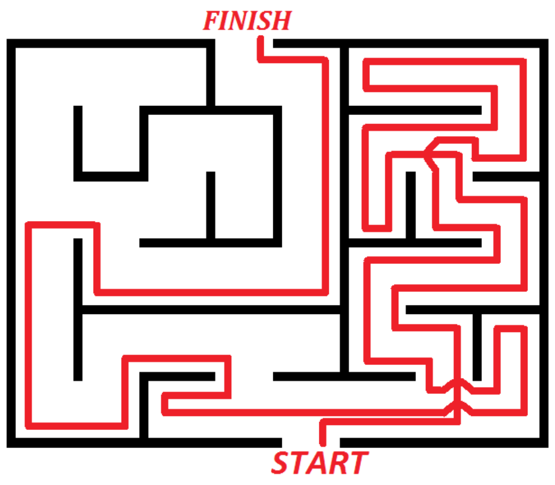


Fig. 3: "Trémaux's algorithm" sample maze solved [1]

The purpose of this project was to develop an algorithm that meets two objectives: provides a complete maze mapping and produces information for generation of shortest paths to exit. The proposed solution combines elements of presented maze exploration algorithms with some novel ideas and is explained in the following sections.

3 Proposed Algorithm

The proposed algorithm combines elements from the two aforementioned methods: "Left Wall Follower" and "Trémaux's algorithm" to ensure both maze exploration and path finding functionalities.

3.1 Algorithm description

The proposed algorithm uses "Left Wall Follower" rule, which says that if it is possible, turn left. As a result, a mobile platform that explores the maze, can do it in a systematic way, which simplifies implementation of the algorithm. The second functionality - path-finding - is provided by the Trémaux method: already visited passages are marked and the information about traversed path is stored in an array A[1..n][1..m] (where n and m defines the size of the maze in the unit of number of cells). As a result, all dead-ends as well as all passages, which lead to the exit, are appropriately identified. Information about all turns made on junctions encountered during exploration of the maze are stored on the stack, which is initialized as empty.

The algorithm can be summarized in the following steps:

Step 1: Explore a maze using "Left Wall Follower" rule and store visited passages in an array A[1..n][1..m], according to "Trémaux's algorithm". If an exit is found, then save the travelled path in a memory.

Step 2: If the whole maze is explored, then set a *flagFull* to 1.

Step 3: If *flagFull* is 1, then choose the shortest path from all stored ones and terminate the procedure, else go to **Step 4**.

Step 4: If *flagFull* is 0, then go back to the first not visited passage.

Step 5: Using "Left Wall Follower" rule explore unvisted part of the maze virtually marking traversed passages.

Step 6: If exit is found, then save a path in the memory and go to **Step 2**.

Step 7: If the start point is found and the stack is empty, then go to **Step 1**.

Step 8: If the start point is found and the stack is not empty, then reverse stack content, appropriately convert values, save the path into a memory and go to **Step 1**.

During the first stage of the algorithm, the first path from the start point to the exit of a maze is found and saved in a robot memory. From the start point information about travelled path is kept in the memory (shape of the path as well as the distance), the traveller is moving forward until a junction or dead-end is met. If a new passage is explored and a junction, which has not been visited before is encountered, new passage is chosen using "LWF" rule. When a dead-end is met, traveller turns around and goes the way he came, remembering that this path led to a dead-end. The same action is taken when on a new passage traveller finds a junction, which was visited before. If a traveller is walking a passage which was visited before, and encounters a junction, a new passage, if available, should be chosen using "LWF" rule. Otherwise, using the same rule, an old passage, but only this which was visited exactly once, can be chosen.

If a maze does not have a solution, a traveller will return to the start point. If an exit of a maze is found for the first time, the second step of the algorithm is executed. A traveller has to check if there are still any unexplored passages, and if so, continues the exploration and explores unvisited branches. Only newly visited passages are marked. Each time when a traveller finds the way out of a maze the new path is additionally stored in the memory.

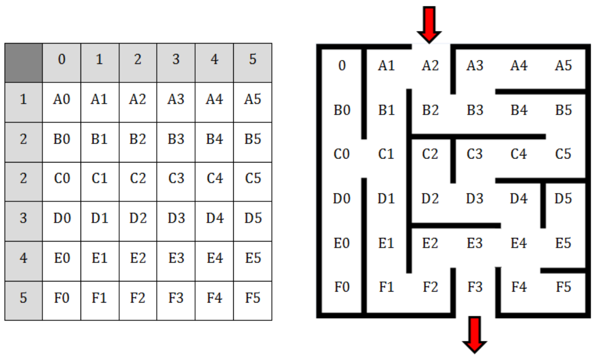
At this point, three different cases are possible. The first scenario expects that the start point is found and the memory containing information about current solution is empty, which means that during backward exploration, new solution was not found. In the second possible situation, start point is met and path representing solution of the maze is stored in the temporary memory. This case is the most complicated, since found solution is leading from the end to the start of a maze, which is opposite to the desired. Performing a few appropriate operations, conversion of the solution is made and saved in the memory as one of possible solutions. The third situation occurs, when traveller founds exit. In this case, full content of the temporary memory is stored as a possible solution.

Finally, when the condition for exploring all places in the maze is satisfied, traveller is going back to the beginning of the maze using path stored in the temporary memory. Afterwards, length of each saved solution can be calculated and the shortest possible path is chosen. In case of maze, which does not have a solution, traveller will come back to the start point, marking all passages as visited twice. Exit will not be found and no path will be stored in the memory.

3.2 Algorithm implementation

To test the developed algorithm, it has been implemented on a small mobile platform. This approach forced a specific way of implementation. Information about a path is saved in a two dimensional array, which represents a structure of the maze. In order to reduce the problem, it was assumed that the starting position as well as a size of the maze is known before entering it. In such a case, there is no need to resize the array during the program execution. Each array cell reflects an exact position in the maze (Fig. 4). Information about all turns made on junctions encountered during exploration of the maze are stored in a stack.

At the beginning the array is filled with zeros. Zeros are equivalent to unvisited cells. Similarly to Trémaux's algorithm, a value of each visited cell is automatically increased by one - a "virtual" line in the maze is drawn. As mentioned before, a traveller is moving in accordance to "Left Wall Follower" rule. It means that if junction that was not visited before is met, traveller chooses the path according to mentioned rule. Whenever a dead-end is faced, the traveller turns around and goes the way he came. Length of a path, which leads to the exit, is measured as a number of travelled cells. Each time, when a new cell is visited, a variable that stores distance is increased by one. Whenever a dead-end is faced, the traveller turns around and goes the way he came, and the length of the path is decreased by one. During that operation, the same path is traversed second time, which also causes increase of the cell values by one. As a consequence, cells that were visited only once, store a value one, while those visited twice (excluded paths) have value two (respectively, in Trémaux's algorithm it would be represented as a double line).



	0	1	2	3	4	5
1	A0	A1	A2	A3	A4	A5
2	B0	B1	B2	B3	B4	B5
2	C0	C1	C2	C3	C4	C5
3	D0	D1	D2	D3	D4	D5
4	E0	E1	E2	E3	E4	E5
5	F0	F1	F2	F3	F4	F5

Fig. 4: Mapping the maze into an array

It can be stated that passages coming out of a junction are prioritized, indicating the order of selection. "Zero" - the highest priority, "One" - a lower priority, "Two" - do not enter. On each hierarchy level, "LWF" rule applies.

The main advantage of an array implementation is the fact that positioning of the traveller becomes very simple. What is more, checking whether a maze was fully explored, requires verification if any "zeros" are left in the array.

Data that is pushed on the stack is a value of an angle, at which the mobile platform turns. Only turns on junctions, from which alternate passages comes out, are pushed on the stack in Fig. 6. Values representing right, left, forward and backward directions are presented in Fig. 5.

Storing subsequent turns made by the traveller, enables

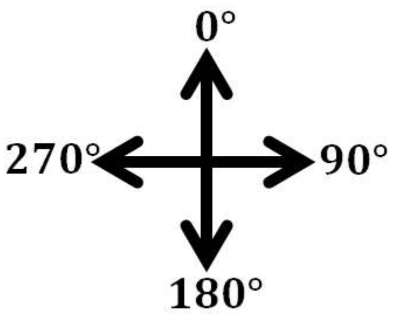


Fig. 5: Values used to describe direction of turns

to restore the traversed path, which in turn, together with mapped array, enables to find the way out of a maze. On the other hand, if a dead-end is found, saved turns on the stack are used to find way back to the last appropriate junction. The corresponding elements are removed from the stack, which enables to keep only information related to a solution. It must be emphasized, that popping data from the stack does not mean that information about explored part of maze, which turned out to be dead-end, are lost. Quite the opposite, this data is stored in the array. There is one special case where the value of the stack must be further modified, which was described in the previous paragraph. If the traveller has explored the maze from the exit to the start and some data is left on the stack, it means that a new solution has been found. In such a case, a contents of the stack must be reversed and each value, except of 0°, has to be increased by 180°, with assumption that if final value will be equal or greater than 360°, then value 360° has to be subtracted.

As soon as the exit is found, a copy of the stack content is saved in the memory. This data, together with values stored in an array, gives all possible maze solutions. From this point, two kinds of stack can be distinguished - solution stack and operating stack. It cannot be forgotten to store also zero degree angles. Although mobile platform does not turn on such a junction, not recording this maneuver, would result in missing junctions on a path from

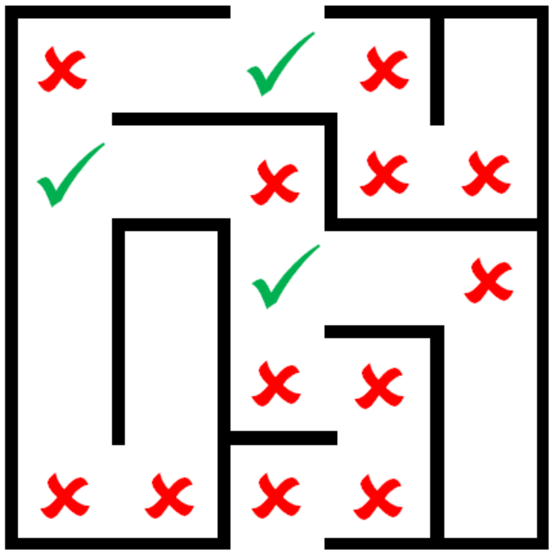


Fig. 6: Maze with marked turns, which would be or would not be stored on the stack

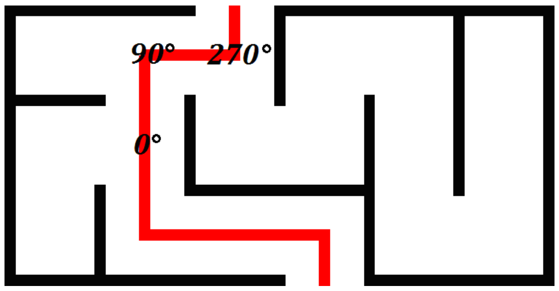


Fig. 7: Marking angles at path junctions

start to the exit. Situation in which storing "forward" turn is necessary, is presented in Fig. 7.

Another important assumption is that if a junction is visited for the second or subsequent time, traveller has to turn to an initial orientation - relative to the maze, in which he was when he has visited the junction for the first time. Only with respect to this position, the next turn can be determined.

Figure 8 presents a filled array after full maze exploration. The content of two stacks which contain all turns leading to the way out of the maze are presented in Fig. 9.

Taking into consideration memory requirements of the algorithm, all information are stored in an array and stack.

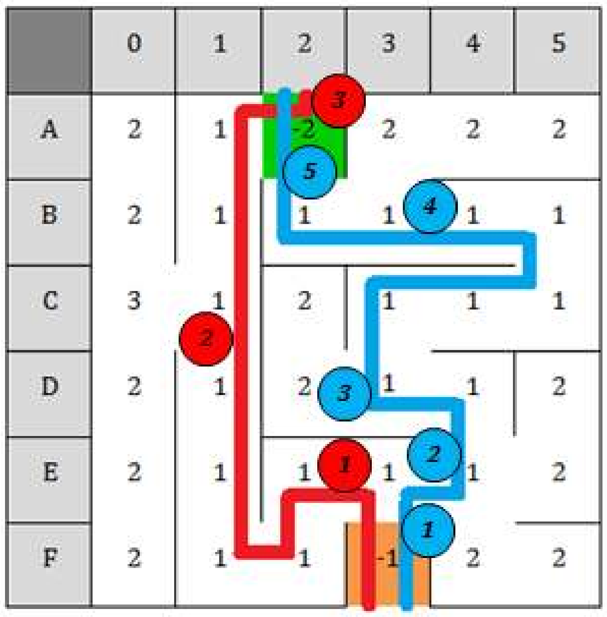


Fig. 8: Filled array after full maze exploration, with two solutions marked

The array is type integer and has the same dimensions as the maze. Its memory size will increase linearly with dimensions of the labyrinth, which in turn enables to approximate required amount of space in the memory to store the whole array. On the other hand, information about turns is stored on the stack of type integer. In this case, it is more difficult to predict required memory. Its size will change with the number of turns, which depends on the complexity of the maze. However, it can be stated that the number of turns will not exceed the number of elements in the array. Possibility of several solutions should be also taken into account. In such case memory for a few stacks has to be reserved.

4 Robot description

A small mobile platform equipped with two DC engines was used throughout experiments. In order to navigate in a maze, information from several sensors: accelerometers, gyroscopes, encoders and ultrasound, was used. Fig. 10

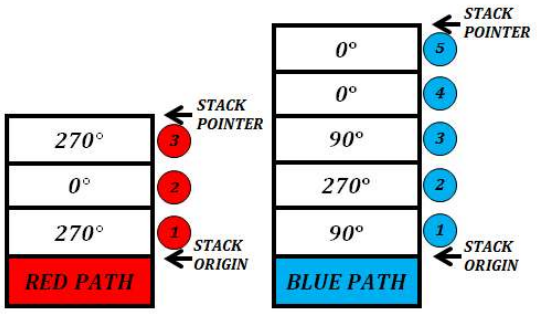


Fig. 9: Two stacks presenting possible solutions

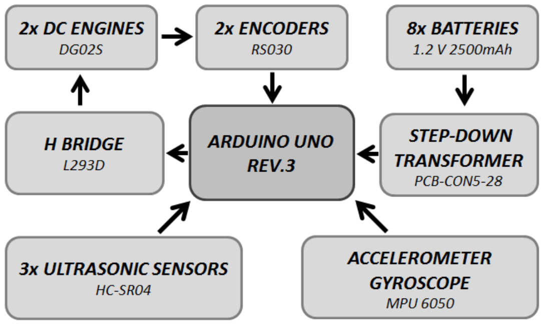


Fig. 10: Block diagram of connections and relationships between components of small mobile platform

presents a block diagram of connections and relationships between components of the platform. Fig. 11 and 12 provide a detailed view of the mobile platform.

All components of the platform are powered by a circuit, which consists of eight 1.2V batteries (Fig. 12 - point 6) connected in series, and a step-down transformer, PCB-CON5-28 (Fig. 12 - point 5). An initial supply volatage of around 9.6V is decreased by step-down transformer to a required level of 5V. Stable input voltage is crucial for proper operation of all used electronic components. Voltage stabilization reduces measurement errors of sensors.

The system can generate 3A current, which is sufficient for driving two DC-motors DG01D (Fig. 12 - point 8), via H bridge L923D (Fig. 11 - point 3). The main idea of controlling both engines bases on Pulse Width Modulation (PWM), with constant amplitude and frequency. By ob-

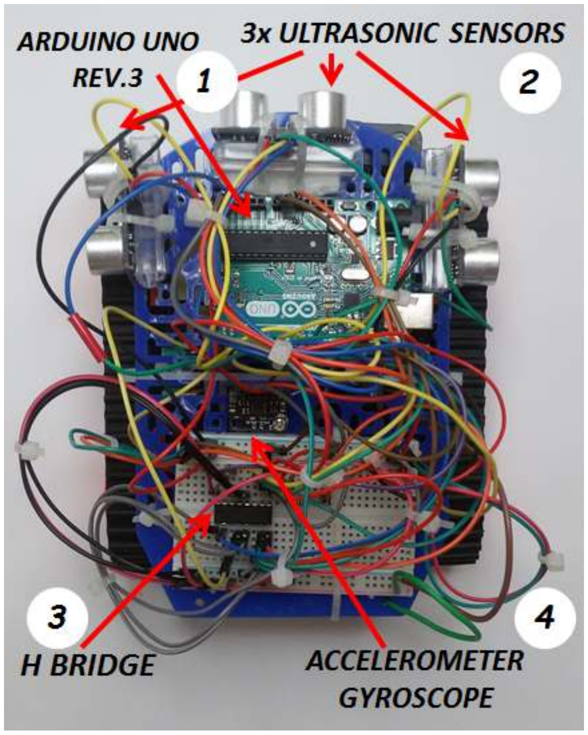


Fig. 11: Small mobile platform components - upper part

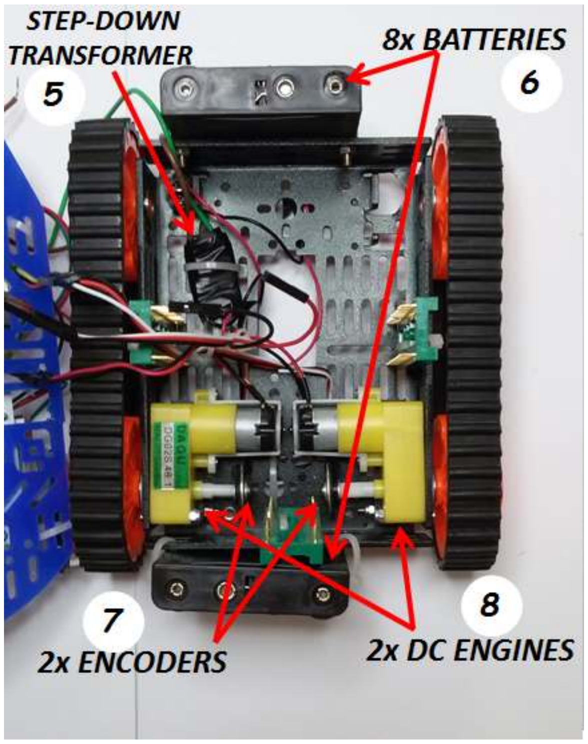


Fig. 12: Small mobile platform components - lower part

taining the desired effective voltage at motors' terminals, smooth regulation of wheels angular velocity is possible. A control signal applied to the bridge is generated using Arduino UNO platform (Fig. 11 - point 1). Such a system, enables programmer to control both wheels seperately, with different speed and direction. As a consequence, it allows to navigate a mobile platform in each direction (forward, backward, left and right).

In order to determine a distance travelled by the vehicle as well as for correct straight track driving, two electromechanical encoders RS030 (Fig. 12 - point 7) are used. Each encoder is attached to one wheel. The device consists of neodymium 4-pole magnets with rubber hub and hall-effect sensor, which allows to determine current position of the wheel. Signals from encoders are sent to Arduino UNO, which basing on counted impulses, after simple calculations, computes linear distance traversed by the vehicle. On the other hand, comparing values from two encoders, it is possible to distinguish whether one of wheels is turning faster than another, which in turn helps to correct the path. At this point it must be emphasized that in order to move straight forward it is not enough to put identical effective voltage for both engines, since each DC engine, will give slightly different angular speed for the same voltage input.

The most sensitive component of the mobile platform is motion tracking device MPU-6050 (Fig. 11 - point 4), which combines a 3-axis gyroscope and a 3-axis accelerometer on the same printed circuit board together with an onboard Digital Motion Processor (DMP) capable of processing complex 9-axis Motion Fusion algorithms [4]. Data collected from MPU-6050 allows to determine angular orientation of the mobile platform in the maze. Having this kind of data, precise rotation of the vehicle at a specified angle (in this case it would be multiple of 90 degrees) is possible. There is one crucial drawback of motion tracking device, such as MPU-6050. Along with the time of collecting data, the measurement error significantly increases. For this reason it was not reasonable to use this data for the purpose of correcting forward motion. Nevertheless, comparing data collected from MPU-6050, obtained in a short period of time gives an opportunity to determine angle of turn with relatively small error. This assumption is sufficient for controlling motion of the vehicle during turns.

The platform is equipped with three ultrasonic sensors HC-SR04 (Fig. 11 - point 2), which provide non-contact measurment function ranging from 2cm to 400cm [5]. The module includes ultrasonic transmitters, receiver and a control circuit. Sensors are mounted on the vehicle in such way that the distance between mobile platform and the nearest wall can be precisely measured from three sides: front, right and left. Algorithm implemented on Arduino UNO, basing on these data, is able to distinguish between free passages and walls. This type of information is directly used to effectively explore a maze, and map its structure in robot memory.

Figure 13 presents constructed small mobile platform in a fully equipped version.

5 Experiments

In order to verify the effectiveness of the presented algorithm, a few experiments were performed. The first part involved computer simulations, aimed to verify logical correctness as well as collect statistical data about algorithm effectiveness. In the second phase of experiments, the developed algorithm was implemented on the platform, set to explore few different mazes and, subsequently, to find the shortest possible paths.

The main purpose of the experiment was to confront performance of the proposed algorithm and the two reference ones. As a consequence, three different maze exploration algorithms were examined:

- "Left Wall Follower",
- "Trémaux algorithm",
- developed algorithm.

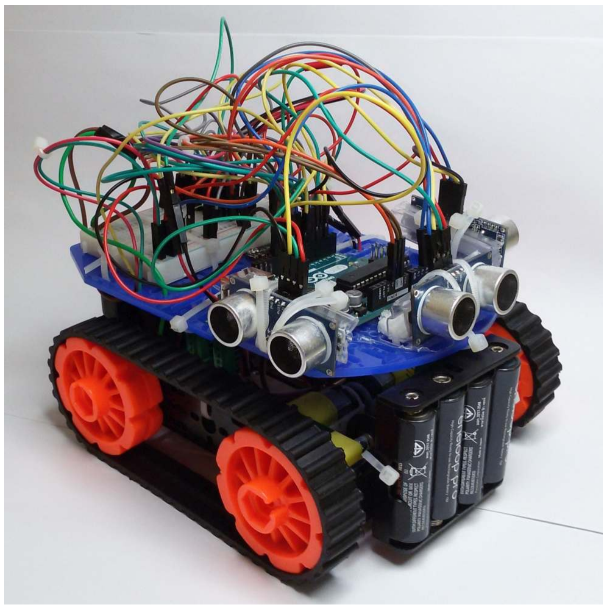


Fig. 13: Final version of self-constructed small mobile platform

5.1 Computer simulations

All three maze exploration methods, were implemented in a form of a C++ program on a desktop computer. Few mazes of different structure and size was prepared and fed as input to the code. Mazes were generated on the website http://www.mazegenerator.net/. The user defines the number of cells (range from 2 to 200), and the parameters E and R (values from 0 to 100). The first one defines the relationship between path length and the size of the maze, while the second affects to the length and quantity of dead-ends. The higher the value, the less branches in the maze, but their path is longer. The mazes of two different size were used in the experiments (small - 50 cells and large - 200 cells). For each size three sets of the labirynths were generated - soft, medium and hard. It should be pointed out that structure of generated mazes excludes free standing walls - not connected with any of the side walls of the maze, which makes "LWF" always

Tab. 1: Length of paths determined for six different maze structures by three algorithms: "Left Wall Follower", "Trémaux's algorithm", Developed Algorithm, using computer simulations

	"LWF"	"Trémaux's"	Developed
Soft Small	5223	804	239
Soft Large	81706	10525	836
Medium Small	5136	728	237
Medium Large	79136	9463	885
Hard Small	5014	684	464
Hard Large	79461	6130	2032

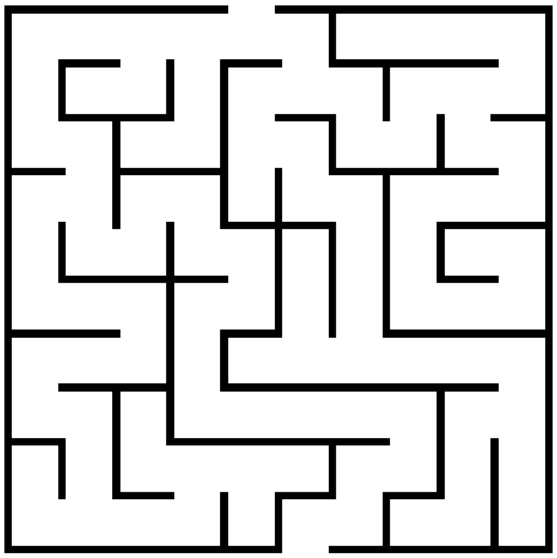


Fig. 14: Sample maze used in computer simulations

capable of finding the way out. All three algorithms were tested on each set of mazes. An exception was the "Trémaux's algorithm", which was run 10 times for each maze and the final result was calculated as an average from all attempts. The aim of the experiment was to find, which algorithm determines the shortest path. Results are presented in Tab. 1. A sample maze is presented on the Fig. 14.

As it can be seen in the Tab.1, the longest path is determined by the "Left Wall Follower" method. It is approximately 4500% less efficient than the developed algorithm. "Trémaux's algorithm" is significantly more effective than "LWF", but at the same time it finds paths leading to the exit 470% longer than the developed algorithm. Computer simulations confirmed that the developed algorithm operates correctly and is significantly more efficient than "Left Wall Follower" method and "Trémaux's algorithm".

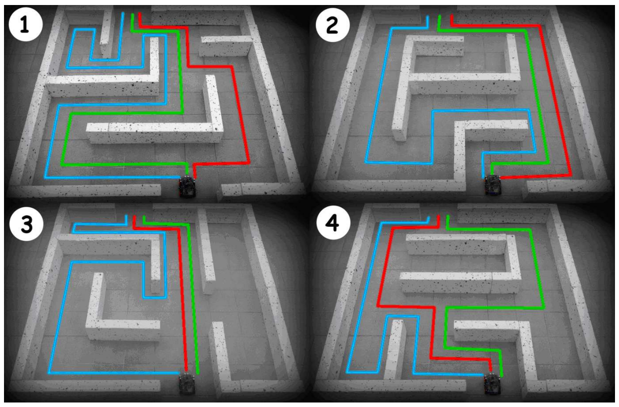


Fig. 15: Four different maze structures taken for tests with marked paths for three algorithms: a) blue - "Left Wall Follower" b) green - "Trémaux's algorithm" c) red - Developed algorithm

5.2 Test stand description - maze exploration by small mobile platform

The next part of the experiment, was to test the developed algorithm in a real environment.

Mazes have been built using polystyrene walls. Each wall is perpendicular or parallel to another and the maze is composed of equal cells of size 60 cm by 60 cm, which simplifies cell mapping in vehicle's memory. Four different maze structures were used (Fig. 12) in testing each of the three algorithms: "Left Wall Follower", "Trémaux's algorithm" and the developed one.

For each maze, a blue path indicates "Left Wall Follower", while a red one shows a path for the developed algorithm and, finally, a green line presents cells traversed by a selected run of Trémaux's algorithm.

Distances traversed by each algorithm for all four presented mazes in the Fig. 15, are shown in Table 2. Similarly as it was in case of computer simulation, for "Trémaux's algorithm", the platform was launched 10 times

Tab. 2: Length of paths determined for four different maze structures by three algorithms: "Left Wall Follower", "Trémaux's algorithm", Developed Algorithm, using small mobile platform

	"LWF"	"Trémaux's"	Developed
Maze No. 1	15	8	7
Maze No. 2	9	8.2	7
Maze No. 3	13	8.6	5
Maze No. 4	9	7.6	7

through the maze. The final path length is calculated as an arithmetic average from all attempts.

Analysing obtained results, it can be stated that developed algorithm greatly improves exploration efficiency of small mobile platform in the maze. Invented logic is about 43% more effective than "Left Wall Follower" and around 20% better than "Trémaux's algorithm".

6 Summary

An objective of the paper was to present an algorithm developed for efficient maze exploration, which can become an alternative for already known solutions. The algorithm explores a whole maze in a systematic way and finds the shortest possible way out. Experimental evaluation of the algorithm shows that it outperforms its counterparts and provides:

- a solution to arbitrary maze types,
- systematic maze exploration,
- determination of the shortest possible way out of a maze,
- can be implemented on mobile platforms with simple microcontrollers.

The proposed algorithm proved efficiency in real life conditions, enabling autonomous exploration of previously unknown environments, avoiding obstacles and finding shortest possible way between chosen points.

References

[1] Anagnostou, L. (2009). Maze Solving Algorithms: Tremaux's Algorithm Visual Example, https://www.youtube.com/watch?v=6OzpKm4te-E

[2] Bares, J.E., Wettergreen, D.S. (1999). Dante II: Technical description, results, and lessons learned. *The International Journal of Robotics Research*, 18(7), 621-649

[3] Durrant-Whyte, H., Majumder, S., Thrun, S., De Battista, M., Scheding, S. (2003). A bayesian algorithm for simultaneous localisation and map building. In *Robotics Research* (pp. 49-60). Springer Berlin Heidelberg

[4] InvenSense Inc. (2012). MPU-6000 and MPU-6050 Product Specification Revision 3.3. *Sunnyvale*, 6–7

[5] ITead Studio. (2010). Ultrasonic ranging module HC-SR04. *Micropik*, pp 1–3

[6] Klein, R., Kamphans, T. (2011). Pledge's Algorithm-How to Escape from a Dark Maze. In *Algorithms Unplugged* (pp. 69-75). Springer Berlin Heidelberg

[7] Mishra, S., Bande, P. (2008, November). Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference* on (pp. 86-93). IEEE

[8] Snapp, R.R. (2010). Threading Mazes, [online access 07.05.2016], http://www.cems.uvm.edu/~snapp/teaching/cs32/lectures/tremaux.pdf

[9] Thrun, S., Thayer, S., Whittaker, W., Baker, C., Burgard, W., Ferguson, D., Reverte, C. (2004). Autonomous exploration and mapping of abandoned mines. *IEEE Robotics & Automation Magazine*, 11(4), 79-91