Journal of Theoretical and Applied Information Technology
15th July 2020. Vol.98. No 13
© 2005 – ongoing JATIT & LLS

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

LOCAL NAVIGATION MAP PROCESSING ALGORITHM TO IDENTIFY INDOOR OBJECTS AND GENERATE MOTION TRAJECTORY FOR ANTHROPOMORPHIC ROBOT

[1]YANA KOSTELEY, [2]DMITRY ZHDANOV, [3]ARTEM BUREEV, [4]LIUDMILA KHOKHLOVA

[1]"Instrument-Making" research laboratory, National Research Tomsk State University, Lenin Avenue, 36, Tomsk, 634050, Russia
[2]"Instrument-Making" research laboratory, National Research Tomsk State University, Lenin Avenue, 36, Tomsk, 634050, Russia
[3]"Instrument-Making" research laboratory, National Research Tomsk State University, Lenin Avenue, 36, Tomsk, 634050, Russia
[4]"Instrument-Making" research laboratory, National Research Tomsk State University, Lenin Avenue, 36, Tomsk, 634050, Russia

ABSTRACT

The article describes an approach to the analysis of a local indoor navigation map derived from a lidar point cloud represented as a projection on a horizontal plane. The article analyzes the applicability of the graph theory and binary image processing methods for structuring and uniting the elements of a noisy and segmental local navigation map layout. On an initial map layout (walls, free space and internal environment objects), stand-alone interior elements are detected, and the points of wall borders are filtered and closed up. Mathematical morphology methods and shortest path and connected component separation algorithms are used to complete these tasks. The authors have developed a trajectory generation algorithm consisting of several turning and straight-line motion commands.

Keywords: *Local Navigation Map, Mathematical Morphology, Flood Fill, Lee Algorithm, Connected Component Separation*

1. INTRODUCTION

Segmenting initial data received from lidars, RGBD cameras and stereopairs is important to ensure the navigation of robotic systems and to map indoor premises. It allows evaluating the dimensions of objects that can be used to develop interaction scenarios. The primary segmentation parameters are dividing an indoor map into separate rooms and detecting walls and internal environment objects for every room.

Detecting indoor objects is the site of special interest for this article. A lot of approaches exist to cope with this task. One of the solutions is the use of multiple lidars. For instance, the authors [1] use two lidars to segment floor and wall points from a point cloud received from a laser scanner. The first lidar scans the floor, the second one – other environment objects. Other methods are based on plane segmentation. It is confirmed with many papers devoted to separating objects or planes on the basis of data sets received from RGBD cameras. Many of these methods involve segmentation only [2-5], while others also imply the classification of objects [6, 7] on detected planes. Special interest should be paid to paper [8], as in this paper environment objects are identified, structured and complemented to create a complete indoor map. In this case [8], individual premises are segmented by time labels (the system stops in the process of map scanning, which signalizes about stoppage in an individual room), then the region growing method is used to identify planes inside this room; their set is divided into vertical and horizontal planes; in their turn, vertical planes are segmented into the walls and planes of indoor premises; the intersecting walls of two different premises are analyzed to segment a door aperture.

Another task was to develop safe trajectory planning algorithms to make the robot move from one space point to another inside premises. The motion of the anthropomorphic robot in a plane space along a curved trajectory is characterized by the following peculiarity: all motion operations come down to a set of turns and straight-line movements. For instance, diagonal motion and sidewise motion are described using the same parameters as turns for a preset angle and onward motion. Therefore, at the initial anthropomorphic robot trajectory planning stage, it is sufficient to compose a list of paired "turn-movement" operations and then to develop a scenario of motion along an optimal trajectory depending on the speed of movement and a set of preferable motion operations for a specific anthropomorphic robotic system. Several motion space representation methods are available to solve this task. The best-known methods are road-map, cell decomposition and potential field [9].

The road-map method implies representing potential robot movements as a connected graph that describes a standardized motion path. The best example of this representation method is the maps of cities, streets, railroad tracks and routes. In this case, motion scenario variability is influenced by a chosen motion edge, while there is no variability within the edge, which cannot be used for the anthropomorphic robot that moves inside indoor premises.

The potential field method is based on the following space representation: all obstacles are marked with a virtual negative charge, while the target point is marked with a positive charge. A negatively charged particle is placed at the departure point. A potential antigradient is considered to be an artificial force applied to the robot. An optimal choice is a path with the minimum artificial force applied [10]. This method allows rather quickly finding an optimal motion trajectory that is well suitable for wheeled robots. In order to develop curvilinear motion scenarios using an available set of movement operations for the anthropomorphic robot, it is necessary to additionally analyze the obtained trajectory and update the algorithm.

The cell decomposition or occupancy grids map method describes space as a set of cells where space point connectivity is determined by the neighborhood of the cells at which these points are located [9]. This representation of spatial data often serves as an interim container to mark obstacles that are dictated by points obtained via lidars, stereo cameras and other sensors. It results from the fact that they can indicate the likelihood of obstacles in a predetermined cell based on the number of points in this cell. The Dijkstra algorithm, A* search algorithm (and its modifications), Lee algorithm (as well as Moore algorithm or wave algorithm) are often used to create a trajectory in the cell decomposition space [11]. The Dijkstra algorithm helps find all shortest paths on a chart (with due account for the weight of every edge). In its turn, the A* search algorithm is used to find an optimal path [12]. The Lee algorithm analyzes the distance to all cells that are free for movements. One of the A* search algorithm modifications, the LIAN algorithm, is interesting from the viewpoint of dividing the path into a set of turning and movement operations [13]. This algorithm limits the displacement of a motion angle in every node of the trajectory that consists of equally spaced nodes. The larger the allowable offset angle is, the more curvilinear the path is and vice versa. The general classification of robot path planning algorithms presented in [14].

Also, there is a class of trajectory plotting algorithms based on intelligent technologies. These methods demonstrate good results, but they are often characterized by a lower execution speed [9]. In this connection, this group of methods was not considered.

High speed of command execution and fast response are important for robotic systems when dealing with big global maps and data streams. Authors propose an approach based on the limitation of an area of interest when dealing with external algorithms. For example, when searching for certain objects using computer vision, separate objects in the room can be selected on a map and subsequently their dimensions, vertical position assessed. Then only the data that corresponds to an area of interest obtained from the detected object is transmitted to a pattern recognition algorithm. Another example of this approach can be full path planning by obtaining local paths inside of each isolated space, where only this specific space boundaries are relevant. In turn, segmentation algorithms for walls and free-standing objects should also have a high data update rate, which ensures an optimal response speed. As mentioned above, in most works, a 3D representation of the space is used to segment objects on a map. As an option to optimize the segmentation algorithm, an approach is proposed for analyzing a two-dimensional representation of a map for the primary segmentation of three-dimensional objects in an image.

Therefore, the overall goal was to examine the approach to the analysis of initial data received from the lidar (and, subsequently, from RGBD cameras and stereopairs) not from the viewpoint of working with 3D planes but in terms of processing 2D projections of obtained data to the horizontal floor plane. A scientific team has studied the applicability of the graph theory and binary image processing methods for structuring data on indoor objects. The team has examined the possibility of filtering and processing the initial layout of walls, room space and interior objects. It is suggested that this segmentation will subsequently allow performing the preliminary classification of a room point cloud into separate structured elements that might later be individually processed with more complex algorithms.

The resulting set of algorithms can serve as a basis for algorithms development that solve specific tasks of a robotic system.

The conducted review shows that there are two types of input data for path planning algorithms: a two-dimensional map with 8 or 4 cell connectivity and a graph. The output data of algorithms are coordinates of the path cells in the case of a two-dimensional matrix, and nodes through which the path goes, in the case of a graph. In robotic systems with a limited set of movement operations, both in terms of the number of available commands and movement parameters (e.g. angle and length of movement), it becomes necessary to plan a path on a two-dimensional map by forming a graph of shortest paths, where the edges of the graph and their relative position satisfy the conditions of available robot commands.

Therefore, the findings were used to study an opportunity of adapting the Lee algorithm to generate a motion trajectory for the anthropomorphic robot to ensure the shortest path of safe curvilinear motion. It is suggested that, as opposed to the methods discussed, adapting this algorithm will allow parametrically controlling the variability of potential motion scenarios, ensuring the fast adaptation of an obtained path for the set of operations available to the robot and evaluating this path from the viewpoint of preferred or non-preferred motion operations in space.

Developed path planning algorithm can be adapted for specific tasks and functional capabilities of a robotic system.

2. LITERATURE REVIEW

Singling out connected components is important for both map analysis and the analysis of binarized images. There are several approaches to solve this problem: recursive, multi- and two-pass algorithms [15]. Two-pass algorithms are often used to ensure high processing speed and lower memory usage rates. These algorithms also have multiple implementation options. The other side of this question is to outline the connectivity of one component with another. Usually, four-connected (cross) and eight-connected (quadrate) components are examined. These components differ in the possibility of diagonal connections between components. This article will describe both types of connectivity depending on the tasks and will utilize a two-pass algorithm with a union-find data structure as a table of equivalence. This structure allows storing and uniting disjoint sets or determining the belonging of elements to one of the disjoint sets [16]. Usually, a union-find data structure is implemented by means of trees to ensure an element instance search rate.

Shortest path search is a fundamental approach to solve the problem of mapping. However, this article examines this class of algorithms not from the viewpoint of navigation but to lay out indoor objects for the purpose of successive by-passing and closing these elements. One of the best solutions in terms of the speed rate and introduction of additional path development conditions is the Lee algorithm. This algorithm implies the performance of forward and backward passes. The forward pass is performed to mark out an area with an algorithm similar to the flood fill algorithm. The only difference is that the distance to the beginning of the motion path is used as a label. The “line” data structure is often used to perform the forward pass. This structure ensures the consecutive processing of elements in the order of emergence. The backward pass consists in looking for the first path from the end point to the start point by using the stack data structure [17].

Quite often, maps created via lidars, RGBD cameras and stereo vision show the borders and parts of objects disjointedly or with false surges. It is convenient to additionally process these data by means of mathematic morphology methods. Mathematic morphology allows deriving important properties from binary images. The basic notions of mathematic morphology are “structural element”, as well as “dilation” and “erosion” operations. The structural element is used to perform operations and represents a two-dimensional binary element with odd dimensionalities. During mathematic morphology operations, the central part of this element is applied onto the pixel under study. The dilation operation consists in using logical multiplication between the structural element and the neighborhood of the point processed when every point has a true value. The erosion operation leaves the pixel under study with its true value, while the true cells of the structural element and underlying neighborhood of the pixel coincide. The combination of these operations allows segmenting different elements of objects, e.g., increasing the connectivity of components or singling out their borders [18].

3. MATERIALS AND METHODS

3.1. Proposed Methodology

The authors implemented an algorithm to construct and segment a local navigation map based on data from the lidar, stereo vision and depth sensor in an application written in the C++ language in the QT Creator programming environment with the use of the MinGW platform. To visualize data, they used the OpenGL library implemented in the cross-platform QT framework. The algorithms were tested on a personal computer with the following features: Intel Core i5-4440 CPU 3.10 GHz, 8 GB RAM.

The authors used a data set received by Dorit Borrmann and Hassan Afzal from Jacobs University Bremen gGmbH, Germany. The data are available in 4TU.Datacentrum at [19]. These data were derived from the following resources available in the public domain [20]. The data (three-dimensional coordinate points) consist of room scans obtained via the Riegl VZ-400 lidar and positions (offsets and angles of rotation in three axes) for each scan received via the odometry system. The indoor premises consist of a room with furniture and other moveable objects as well as part of a corridor. Scanning was limited to one story. In addition to the side walls, the data include the images of the ceiling and the floor.

3.2. Algorithm

A. Pre-processing

The article describes an algorithm to segment a local indoor navigation map and does not consider the method used to obtain this layout. The repository data layout [19] used in this article was obtained by means of the following operations:

1) constructing an occupancy grid map with the cell size of 5x5x5 cm;

2) analyzing the layers of the occupancy grid map from the bottom upwards by the number of marked cells in each layer, singling out the level of the ceiling, interval located below the ceiling (including walls) and interval located between the ceiling and the floor (furniture interval);

3) transferring the wall and furniture intervals into two-dimensional representations M_w and M_f;

4) mapping two-dimensional wall interval representation M_w by means of vertical and horizontal passes up to obstacles, uniting the results of these passes via the logical multiplication procedure with the construction of a resulting two-dimensional map of the walls in the current premises, saving the layout of the passage area as array M_w (obstacles are marked with value 1, free space – 2, virgin area – 0);

5) mapping free space by means of the flood fill algorithm (value 3) on array M_f with the construction of a resulting two-dimensional map of the free space in the current premises that is copied to Mf;

6) uniting layouts 1 and 2 of array M_f and layout 3 of array M_w; the layout of the passage area (value 2) obtained at stage 4 and transferred to the current map without overlapping with layout 3 will contain the layout of furniture.

The consecutive use of the mathematic dilation and erosion morphology operations allows joining the isolated elements of interior objects (value 2). In this case, the authors used a chain of the above mentioned operations with structural element 3x3 where the angle units are zero. It allowed constructing a map shown in figure 1.

Then two-dimensional array M_r was created. This array contains the layout of the current room. It will be used to represent the output of the initial algorithm. Therefore, array M_r will store the following layout values: 0 – virgin area; 1 – layout of wall borders; 2 – layout of furniture; 3 – layout of free space for movements within the room.

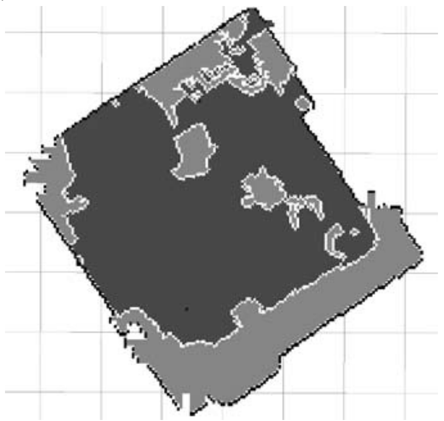
Figure 1: Two-Dimensional Map of Room Layout M_r

B. Connected furniture component separation

A two-pass four-connected components search algorithm was used to segment separate (connected) furniture objects. The first pass consisted in the line-by-line analysis of each filled (value 2) cell of matrix $M_r(x_i,y_j)$ and its neighbors $M_r(x_{i-1},y_j)$ and $M_r(x_i,y_{j-1})$ with due account for the following situations:

- both neighbors have marks, current cell $M_r(x_i,y_j)$ is filled with the minimum value of the neighbors' marks, connection between marks $M_r(x_{i-1},y_j)$ and $M_r(x_i,y_{j-1})$ is indicated in the table of equivalence, if these marks are not equal;

- one of the neighbors has a mark, current cell $M_r(x_i,y_j)$ is labeled with its mark;

- none of the neighbors has a mark, current cell $M_r(x_i,y_j)$ is labeled with a new mark in compliance with the order of passing, and a new mark is added into the table of equivalence (the starting value of the marks is 5).

The union-find data structure was used to create the table of equivalence. During the backward pass, all the marks of the matrix are updated with the minimum value from the table of equivalence. Also, during this pass, information on the number of cells in each connected component, the start point (the first instance of the element of the connected domain in case of line-by-line scanning) of each connected component and its area is collected. This information can be useful for further segmentation and object specification. The mapping result is shown in figure 2.

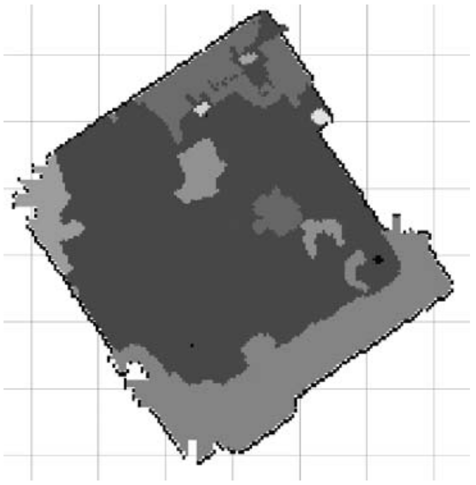
Figure 2: Detecting Connected Components of Furniture

C. Room borders closure

Figure 2 shows that the border layout (mark 1) has an unclosed structure and significant surges. Therefore, the key idea of this algorithm was to close the chain of room wall points. It was suggested that the shortest path would "cut" the surges upon closing the chain of points. In order to obtain a closed curve of room borders, the authors used the mathematic morphology operations and Lee algorithm.

At first, all the marked cells (non-zero values) were copied to buffer matrix M_H. Then buffer matrix M_H sequentially underwent the dilation operation with two structural elements: 5x5 element with completely colored cells and 3x3 element with zero angle units. The results were copied to buffer array M_e. The erosion operation was performed twice with structural element 5x5 and completely colored cells. Matrix M_e was subtracted from array M_H. A heavy room border was obtained as the result of these operations. In order to speed up the algorithm, these operations can be combined with the dilation and erosion operations performed in relation to furniture objects.

The selection of structural elements for the mathematic morphology operations depends on the degree of the noise pollution of data determining borders. Structural elements can be chosen for every specific type of data sets. This border is demonstrated in figure 3. As shown in figure 3, some parts of the border were removed. These objects can be analyzed later. For instance, the surges in the corners of the room can be ignored, as they are unapproachable, while the points displaying lighting fixtures can be segmented and analyzed from the viewpoint of motion restrictions.

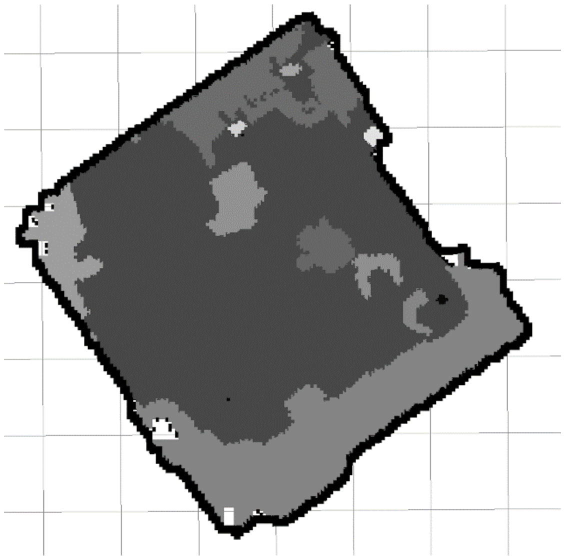
Figure 3: Dilation and Closure of the Room Border Via Mathematic Morphology Operations

This border can be used to create the shortest path and, by that, number the sequence of points the chain of which closes the room borders. It allows obtaining data in a form that is convenient for analyzing room parameters, specifically, a serial chain of border points that can be easily processed by means of approximation and interpolation algorithms, the Hough Transform and other methods to determine curve characteristics.

In order to perform further operations, the following values were entered into matrix M_{Lee}: "-1" – room borders from array M_H, "-5" – virgin space from array M_r (mark "0"), "-4" – room space: free space and furniture (marks "2" and "3" from array M_r). Then all the cells with value "-1" in matrix M_{Lee} were marked with a two-pass eight-connected components search algorithm. The difference between this method and the procedure that was previously performed on furniture objects in array M_r is that the neighbors of $M_{Lee}(x_i,y_j)$ will be not only $M_r(x_{i-1},y_j)$ and $M_r(x_i,y_{j-1})$ but also $M_r(x_{i-1},y_{j-1})$ and $M_r(x_{i+1},y_{j-1})$ (eight-connected neighbors). It will allow considering the components where the cells have neighbors in the diagonal direction to be connected ones. Also, in this case, new components are numbered with start value "-6" and counter decrementation rather than incrementation. During the final run, the list is supplemented not only with information on the number of cells in each connected component and its start point but also with information on the area where it is located (maximum and minimum point coordinates by rows and columns).

As room borders had a closed chain, and launching the Lee algorithm on these data would make it difficult to determine the closing point, the authors decided to open the area of each component. In order to do this, they used information on the location area of the connected component, more specifically, this area was horizontally divided into two halves based on information on the rectangular area where it was located. Switching from mark "-5" or "-4" to the marks of the current connected component index and vice versa was registered from top downward on the border of intersection. If the number of such "switches" was equal to or more than two, this connected component was considered to be a room border, otherwise it was marked as a furniture object. The first "switch" was labeled with mark "-2," mark "-3" was used to label eight-connected neighbors of all the cells with mark "-2" on the left ($M_{Lee}(x_{i-1},y_{j-1})$, $M_{Lee}(x_i,y_{j-1})$, $M_{Lee}(x_{i+1},y_{j-1})$). Therefore, parallel opening lines were created during the first "switch." As it has already been mentioned, the first line with mark "-2" would ensure movements to pass over the chain of the connected component's border to one side (in this case, in a clockwise manner), the second line with mark "-3" would ensure the identification of the end of path and ambiguity of the eight-connected analysis of neighbors. It is noteworthy that dividing the area of the connected component into two halves vertically or adding a constraint line on the right would not change the essence of this algorithm, it would only change the direction of cell scanning.

Before launching the Lee algorithm, a list to save the coordinates of the "end of path" cells is created.

In this case, the Lee algorithm has the following format:

1) eight-connected neighbors of all the cells with mark "-2" on the right ($M_{Lee}(x_{i-1},y_{j+1})$, $M_{Lee}(x_i,y_{j+1})$, $M_{Lee}(x_{i-1},y_{j+1})$) are added into a line and marked with start value "1";

2) while this line is not empty, the Lee algorithm marks all the cells with the value of the current connected component index; if the current cells has a neighbor with mark "-3," this cell is added into the list of the "end of path" cells;

3) the coordinates of the cell with the minimum value of $M_{Lee}(x_i^{end},y_j^{end})$ are derived from the list of the "end of path" cells;

4) a path from this point to cell with mark "1" $M_{Lee}(x_i^{start},y_j^{start})$ is laid by means of a backward pass and stack; if there are several options for the next step, preference is given for the step that has a neighbor with mark "-4," as it allows making a path bordering on furniture or free room space;

5) points $M_{Lee}(x_i^{start},y_j^{start})$ and $M_{Lee}(x_i^{end},y_j^{end})$ are connected via marks "-2" and "-3."

This path can be transferred into matrix M_r and saved as a list for further convenient processing. The result of this algorithm is shown in figure 4.

This result can be used to separate wall sections on depth maps that can be easily transferred to RGB image sections. It will allow pattern recognition algorithms to ignore the sections that are unimportant to recognition. The closed nature of the room borders allows localizing obstacle points spaced within the premises. Also, the border data storage structure ensures a linear pass across all the points, which allows optimizing the speed of by-passing to complete external tasks.

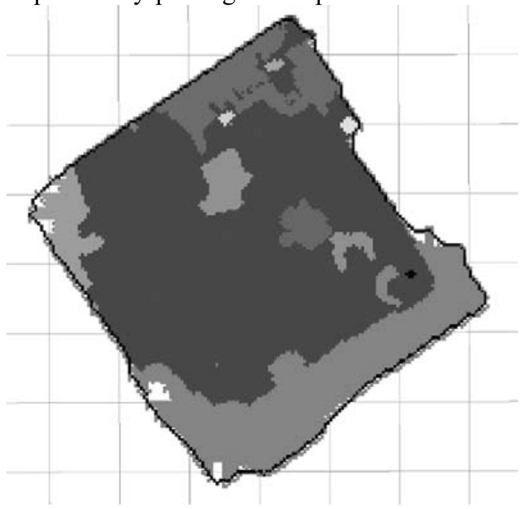
Figure 4: Result of Algorithm

D. Robot trajectory plotting

The following aspects should be considered during robot trajectory plotting:
- a motion scenario consists of a set of movement and turning operations;
- a potential travel distance should be larger than D_{min};
- a movement operation should not exceed D_{max} to control motion;
- it is necessary to generate a trajectory that will minimize the probability of crashing into detected environment objects;
- in addition to the covered distance, the number of turning operations can be used as a criterion to generate an optimal trajectory.

Matrix M_r is copied to new matrix M_t to store the motion trajectory layout. To comply with the safe path criteria, it is necessary to add sections where the robot can crash into obstacles if it gets on the points of these sections. Also, it is important to make the robot be able to turn around its radial axis in every point of the trajectory, as it will allow the robot to maneuver to prevent crashing into moving objects. In this connection, all M_t points located farther than value d_{save} from the closest obstacle are used as a safe zone. This value is calculated in the following way:

$$d_{save} = \max(\frac{w_{robot}}{2}, \frac{h_{robot}}{2}) + d_{extra} \quad (1)$$

where w_{robot} - robot width (mm);

h_{robot} - robot height (mm);

d_{extra} - value of the robot's potential deviation from its current location point as it makes a turn.

Hereinafter in the description of this algorithm, let us assume that value d_{save} equals 150 mm. This value will allow more completely demonstrating the operation of this algorithm with a test data set [19].

To exclude the sections marked as unsafe zones within the robot's motion area, a set of marks was introduced within formed matrix *Mt*:
- mark "2" – a zone free for motion;
- mark "3" – a zone attributed to furniture inside the room;
- mark "1"- a zone characterized as furniture borders.

When the algorithm is executed, a neighborhood is generated for every point marked as "1" and "3." This neighborhood represents a circle where the center is located in the analyzed point. The circle radius is defined with value d_{save}. All marks "2" located within this circle are identified as unsafe and labeled with mark "-1". Marks "2" that do not comply with this condition are labeled with mark "-2."

To create a template determining the offset of all points located inside the circle in relation to its center, the simplest circular formula was used. In the parametric form, it is written as:

$$\begin{aligned} x(R,\varphi) &= R\cos(\varphi) \\ y(R,\varphi) &= R\sin(\varphi) \end{aligned} \quad (2)$$

where $x(R,\varphi)$, $y(R,\varphi)$ - offsets along *x* and *y* from the center of the circle with radius *R* for the points located on the border of this circle when positive axis ox is turned by angle φ.

Let us name this template the "impassable area mask." Therefore, the "impassable area mask" includes all *x* and *y* values that are unique for the multitude of offset coordinates and calculated from the formula (2) for parameter ranges $R=[0^0,359^0]$ and $\varphi=[1, d_{save}]$. This template can be used to determine an area the robot will occupy when making standing turns (figure 5).

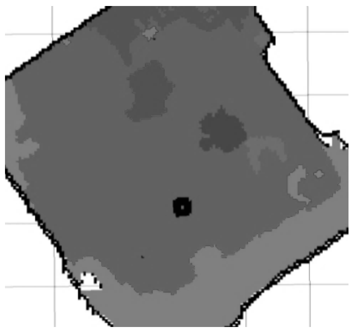

Figure 5: Robot's Turning Area

In this case, *matrix M_t* has the look shown in figure 6. Marks "-2" (gray color) indicate a path that is free for motion, while marks "-1" indicate unreachable zones (black color), and marks "-3" indicate the space outside the room (transparent sections).

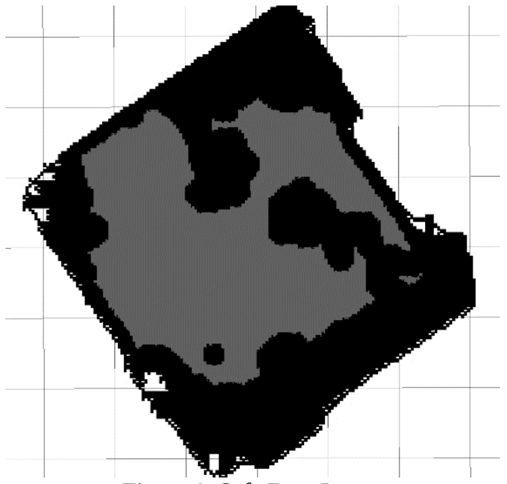
Figure 6: Safe Zone Layout

Then the start point of the trajectory is marked within matrix M_t, and the shortest paths for the robot to every point of the safe zone are mapped out from this start point. In order to increase the processing speed, it is possible to preliminarily set the end point of the trajectory. When this point is reached, it is necessary to stop mapping-out the regions where the reachability index (minimum number of cells that can be passed through to get to this point from the start point) is higher or equal to the reachability index of the trajectory end. This condition is especially important to maps having considerable dimensions. As for the data described in the article, the execution speed did not reduce, as the start and end points chosen to demonstrate the functionality of the algorithm were located far from each other (figure 7).

Then it is necessary to map out the sections of matrix M_t where the reachability index will be minimum when the robot moves from the start point to the end point. Hereinafter these sections will be referred to as trajectory regions. To do this, the authors used the depth-first search algorithm with a stack as opposed to the previous stage where the width-first search algorithm with the “queue” data structure was used. Also, logic (boolean) array M_{tb} is formed in advance. This array stores a flag indicating that the current cell belongs to the trajectory zone. As a type of elements included into the stack, the authors used a structure with the following fields: cell coordinates, cell reachability index and pointer to the element from which the path to this cell was traveled. To optimize memory utilization, every neighborhood element under consideration was included into “map” data structure m_{tz}, so a pointer to this element in “map” m_{tz} was included into the stack or into the pointer to the previous element.

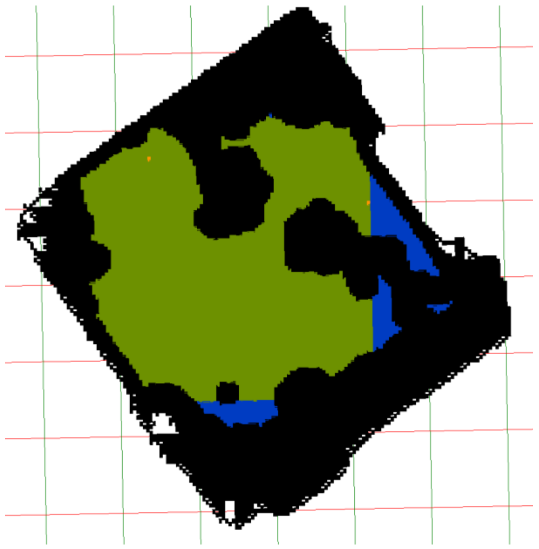
Figure 7: Safe Zone Layout Based on The Lee Algorithm

During the execution of the described algorithm, depth-first search was performed at the initial stage. It ensured mapping out matrix M_t from the end point to the start point of the path. Then matrix M_{tb} ensuring trajectory restoration (it is not performed if the start point of the trajectory is reached) was mapped out.

An informal description of the trajectory variation generation algorithm can be formulated in the following way:

- the end element is included into the stack;
- subsequent operations are being performed while the stack is not full;
- if the current element is not marked in M_{tb}, and it is not the start point of the trajectory: let us examine the eight-neighborhood of this point. If the reachability index of a neighborhood point is one point lower than the current element, and it is marked in M_{tb}, the current element is marked in M_{tb}; if it is not marked, the current element and this neighborhood point are included into the stack (indicating that we have reached this neighborhood point from the current element);
- if the current element is marked in M_{tb}, or this is the start point of the trajectory, and there is a pointer to the previous element, it is included into M_{tb};
- the algorithm moves to the next iteration of the stack cycle.

The result of this algorithm is shown in figure 8.

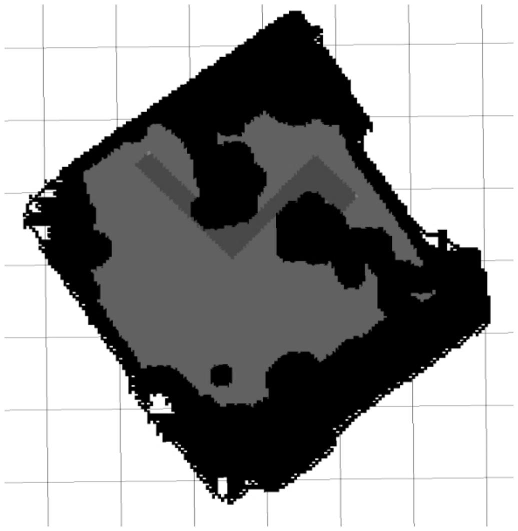

Figure 8: Array M_{tz} Mapping Out

Then, for every mapped-out point of matrix M_{tb}, it is necessary to calculate the number of eight-neighborhood elements that are not mapped out in matrix M_{tb}. To do this, it is sufficient to pass all mapped-out elements in matrix M_{tz} (figure 8) and analyze their neighborhoods. Also, in matrix M_t, all the points with a positive value (Lee algorithm) and without a signaled state in a corresponding cell of matrix M_{tb}, are marked as "-1."

Then a set of turning and movement operations is formed from the layout obtained inside matrix M_t.

Let us analyze figure 9 and imagine that a decision on generating a motion trajectory for the robot is made by a human. Let us mark the start of the robot's path as "*" and the end of the robot's path as "?." It is obvious that the following options would be considered in this case: from "*" to "+," and then to "?," or from "*" to "-," and then to "?."

When a path from "*" to "^" is chosen, the number of turning operations is increased, as it is necessary to move from "^" to "19" and travel to "?" only after this.

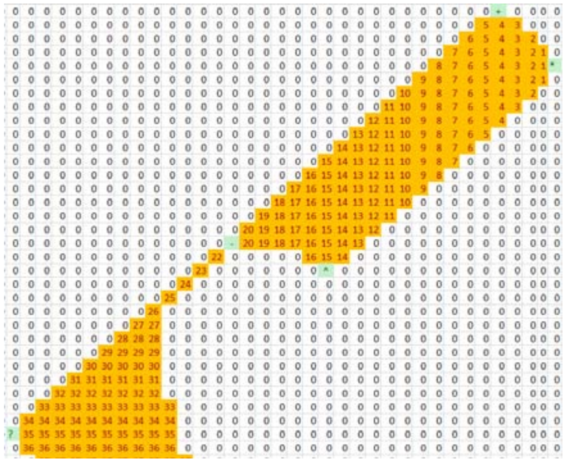

Figure 9: Array M_{tz} Layout

Also, it is possible to use point "*" to get to any point on the line from "+" to "-." It will require the same number of turns as when moving through points "+" and "-," but it will lead to the higher variability of solutions that are practically identical.

If one attentively examines figure 9, one will easily notice the following regularity: points "+," "-," "?" and "^" fit the same condition that can be formulated in the following way. A point is "terminal" if it has five virgin neighbors or it has the maximum possible straight-lined deviation from the previous step in the trajectory zone. The presence of six virgin neighbors implies that there is only one possible continuous straight-lined path through this point (elements "22"-"25" in figure 9).

Therefore, in order to create a trajectory plotting algorithm, it is necessary to define a mechanism for verifying the existence of a straight-lined path of length D through the mapped-out region from point (x_c,y_c) to a virgin area with a robot's turn by angle φ. To achieve this, the authors formed a data structure containing the following fields: an offset *(dx;dy)* along axes *x* and *y* against the current zero point *(0,0)*, a set of angles and distances from which this point can be reached, a length of the line segment from the zero point to point *(dx;dy)*. A combination of unique points that define the neighborhood of the point in the circle of radius D_{max} is included into a data set consisting of the elements belonging to the structure described above – point neighborhood or trajectory node. The points located in the neighborhood of the trajectory point should be normalized to the value of cells in M_t. Value D_{max} defines the maximum straight-lined path the robot can cover to the target point. If value D_{max} reduces, it suggests that the number of trajectory nodes increases. If value D_{max} is optimal, it allows covering all the longest straight-lined paths within mapped-out region M_t. If value D_{max} increases, the size of the trajectory point neighborhood will also increase, which will require increasing the turning angle by a lower value when forming this neighborhood.

Then it is necessary to examine all possible optimal motion scenarios. Straight-lined motion from this node is considered optimal if it longest and leads to a point that has five virgin eight-neighbors. In addition, it is important that only one longest straight-lined motion is defined under the current node, and several motions can be considered towards the point with five virgin eight-neighbors. For this purpose, the authors used buffer $Node_{Max}$ for the longest motion and list $Node_{Opt}$ for the motions to the point with five eight-neighbors.

The algorithm should not consider straight-lined motions where the end points can be used to generate a path by other straight-lined motions from the same node. That is, if a straight-lined path from the current node can be generated above a point, and this point is not terminal for this straight-lined motion, it cannot be the end of a straight-lined path for other motions from this node. In this connection, a passability matrix is formed to ensure the operation of the algorithm where such a condition is met for every node analyzed.

Other straight-lined motions from the current node are not considered if a straight-lined path to the end of the trajectory (end point of the robot's path) already exists from this node. Such a node is marked as the end of path.

Also, there is no need to consider straight-lined motions with the same end point for the node under study. For this purpose, the authors created a "map" of unique potential nodes that can be reached from the node under study.

To restore the path, all the end nodes of the trajectory (that reached the end of path) are included into the list of the end of path. For every node, information on the parent node from which it was reached is saved. Inside the node, there is a data structure that keeps a trajectory point neighborhood element determining the turning angle and length of the straight-lined motion necessary to reach this node from the parent node.

In this connection, the authors created the following algorithm:

- place a start point to a node, specify the robot's current turning angle, and place the node into a queue;
- if the queue is not empty, take the next node in the order of priority, otherwise complete the algorithm;
- if the path ends in this node (the target point is reached), put this node into the "end of path" list;
- if the current node is detected on the path with the number of nodes that is significantly higher than that of the already obtained destination path, move to the next iteration of the cycle;
- initiate a passability matrix, buffer $Node_{Max}$, list $Node_{Opt}$, end of path flag, "map" of unique potential nodes for motion from this node;
- launch two cycles: an external cycle – movement within the turning angle, and an internal cycle – movement along the straight-lined path;
- take a trajectory point neighborhood element that fits the current turning angle and straight-lined motion defined by the cycle parameters;
- if this movement goes beyond the boundaries of matrix *Mt*, move to the next iteration of the external cycle;
- if the Lee algorithm mark (in matrix *Mt*) is lower in the movement point vs. the current node, move to the next iteration of the external cycle;
- if the mark of the current movement point is defined, and D_{max} is not reached, record this point in the passability matrix and move to the next iteration of the internal cycle;
- if the end of path is reached with this movement, add this node into the queue, put an end of path flag and exit from the internal and external cycles;
- if this movement reaches a virgin area in matrix *Mt* (mark "-1"), calculate the previous movement of the internal cycle;
- if this movement reaches a virgin area, or D_{max} is achieved, check the availability of five virgin eight-neighbors in this node. If this condition is met, add this node into array $Node_{Opt}$ and include it into the "map" of unique potential nodes for motion from this node. If buffer $Node_{Max}$ contains a node with a lower value of mark M_t, add the value of this node into buffer $Node_{Max}$. If buffer $Node_{Max}$ contains a node with the same value of mark M_t as that of the current node, leave the node with the minimum vector value in $Node_{Max}$. If this node is added into $Node_{Max}$, register it in the "map" of unique potential nodes for motion from this node. The nodes already included into the "map" of unique potential nodes for motion from this node are not added into $Node_{Opt}$ and $Node_{Max}$;
- if the end of path flag is not marked, put all the nodes from $Node_{Max}$ and $Node_{Opt}$ into the queue, if the node is not added into the passability matrix;
- move to the next iteration of the queue cycle.

The execution of this algorithm for the selected points of the generated map resulted in four paths: two five-node paths, one six-node path and one seven-node path.

Two optimal paths are shown in figures 10-11.

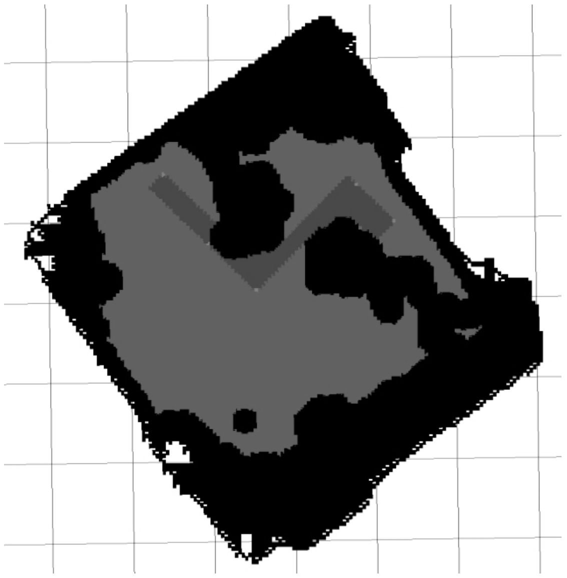
Figure 10: Safe Zone Layout, Optimal Path #1

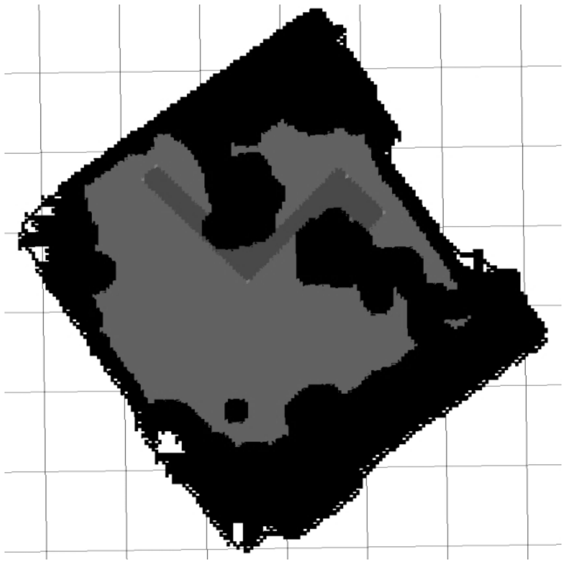
Figure 11: Safe Zone Layout, Optimal Path #2

The trajectories can be easily restored from the list of end points, while the nodes with the minimum number of nodes will be stored at the very beginning of this list.

4. RESULTS AND DISCUSSION

The article describes the local navigation map of premises having the area of 965x880 centimeters (matrix with the dimensions of 193x175 elements) obtained by means of data processing [19]. The algorithm output is shown in figure 4 and 10-11. It represents a local navigation map divided into separate interior objects, free space for motion, connected room borders described with a list containing a sequentially closed chain of points, trajectory plotting. The processing speed of the algorithm stages for this data set is shown in table 1.

Table 1: Processing speed of algorithm stages

	Stage	**Worst execution time, ms**	**Mean execution time, ms**
1	Dilation and erosion of furniture and room components	2	1.5
	Subtraction of erosion results from the results of room components dilation	1	0.2
2	Looking for the connected components of furniture	2	1.3
3	Looking for the connected components of borders	1	0.3
4	Looking for the shortest path	1	0.2
5	Safe zone template	29	28.8
6	Safe zone layout	3	3
7	Wave algorithm	2	2
8	Trajectory region layout	1	1
9	Neighborhood template	407	404.3
10	Trajectory scenario generation	2	1.8

It is noteworthy that, when using the data set [19], the number of connected border components at stage 3 was equal to 1. It raises the following question: under what conditions is it possible to single out a connected border component that will describe the room border in the presence of several connected components? Such a case can be connected with the presence of columns or installations inside the room. In order to solve

this issue, it can be limited to solving the point in polygon problem [20]. It is obvious that every point where a capturing device (lidar, RGBD camera or stereopair) is located will lie within a polygon that describes room borders. At stage 4, all the points of the connected component border are in fact the vertices of the polygon, which easily fits into the initial data of this task.
Stages 5 and 9 can be loaded once upon the launch of the application, and their execution speed is not crucial for the algorithms in general.

5. CONCLUSION

This paper has demonstrated the successful implementation of the algorithm used to segment a local indoor navigation map based on the data set [19]. The algorithm allows detecting the connected components of furniture, closing and opening room borders and representing them in the form of a traversal sequence by the points of the border on a local navigation map containing room borders and marked furniture objects.

The data obtained by means of this algorithm can be later used to identify and segment furniture objects, determine room parameters and develop scenarios for interaction with objects and navigation within this room.

The trajectory obtained by means of these algorithms can be used to directly execute a list of commands or to transform it into diagonal, sideway movements and other operations that can be defined with the turns and linear motion of the anthropomorphic robot.

REFRENCES:

[1] G. Ajay Kumar, Ashok Kumar Patil, Rekha Patil, Seong Sill Park, and Young Ho Chai, "A LiDAR and IMU Integrated Indoor Navigation System for UAVs and Its Application in Real-Time Pipeline Classification", *Sensors*, Vol. 17, No. 6, 2017, p. 1268.

[2] R. Finman, T. Whelan, M. Kaess, and J. J. Leonard, "Efficient incremental map segmentation in dense RGB-D maps", *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[3] A. Morar, F. Moldoveanu, L. Petrescu, O. Balan, and A. Moldoveanu, "Time-consistent segmentation of indoor depth video frames", *40th International Conference on Telecommunications and Signal Processing (TSP)*, 2017.

[4] E. Che and M. Olsen, "Lidar Point Cloud Segmentation", *GIM International,* 2019. [Online]. Available: https://www.gim-international.com/content/article/lidar-point-cloud-segmentation. [Accessed: 09-Feb-2020].

[5] Zhe Zhao and Xiaoping Chen, "Building temporal consistent semantic maps for indoor scenes", *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[6] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images", *Lecture Notes in Computer Science*, 2012, pp. 746-760.

[7] S. Tang, Y. Zhang, Y. Li, Z. Yuan, Y. Wang, X. Zhang, and W. Wang, "Fast and Automatic Reconstruction of Semantically Rich 3D Indoor Maps from Low-quality RGB-D Sequences", *Sensors*, Vol. 19, No. 3, 2019, p. 533.

[8] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms", *Pattern Recognition*, 70, 2017, pp. 25-43.

[9] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory Planning in Robotics", *Mathematics in Computer Science*, Vol. 6, No. 3, 2012, pp. 269-279. DOI: 10.1007/s11786-012-0123-8.

[10] T. Tsuji, P.G. Morasso, and M. Kaneko, "Trajectory generation for manipulators based on artificial potential field approach with adjustable temporal behaviour", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*. DOI: 10.1109/iros.1996.570811.

[11] P. Zwierzykowski, M. Glabowski, P. Nowak, and B. Musznicki, "Review and Performance Analysis of Shortest Path Problem Solving Algorithms", *The International Journal on Advances in Software*, Vol. 7, No. 1 & 2, 2014.

[12] C. Jinhyung and Z. Bo, "The shortest path from shortest distance on a polygon mesh", *Journal of Theoretical and Applied Information Technology*, Vol. 95., No.18, 2017, pp. 4446-4454.

[13] A.I. Panov and K. Yakovlev, "Behavior and Path Planning for the Coalition of Cognitive Robots in Smart Relocation Tasks", *Robot Intelligence Technology and Applications*, Vol. 4, 2016, pp. 3-20. DOI: 10.1007/978-3-319-31293-4_1

[14] M.N. Rastgoo, Mohammad, N. Naim, N. Bahareh, F. Mohammad, N. Ahmad, and Z. Mohd, "A critical evaluation of literature on robot path planning in dynamic environment",

Journal of Theoretical and Applied Information Technology, Vol. 70, No.1, 2014, pp. 177-185.

[15] C. Fiorio and J. Gustedt, "Two linear time Union-Find strategies for image processing", *Theoretical Computer Science*, Vol. 154, No. 2, 1996, pp. 165-181.

[16] M. Nosrati, R. Karimi, and H.A. Hasanvand, "Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches", *World Applied Programming*, Vol. 2, No. 4, 2012, pp. 251-256.

[17] H.J.A. Heijmans and C. Ronse, "The algebraic basis of mathematical morphology I. Dilations and erosions", *Computer Vision, Graphics, and Image Processing*, Vol. 50, No. 3, 1990, pp. 245-295.

[18] D. Borrmann and H. Afzal, "3D scans taken around the Automation Lab at Jacobs University Bremen", *4TU.Datacentrum*. DOI: https://doi.org/10.4121/uuid:066f1025-5151-4d28-8405-954026e584b0

[19] D. Borrmann and H. Afzal, "Robotic 3D scan repository", Universität Osnabrück, Tech. Rep., 2011. [Online]. Available: http://kos.informatik.uni-osnabrueck.de/3Dscans/ [Accessed: 09-Feb-2020].

[20] C.-W. Huang and T.-Y. Shih, "On the complexity of point-in-polygon algorithms", *Computers & Geosciences*, Vol. 23, No. 1, pp. 1997, pp. 109-118.