Construct the Path Around Obstacles Detected by Stereo Vision

Aleksandr Smirnov, Artem Bezzubtsev, Igor Tishchenko
Program Systems Institute of the Russian Academy of Sciences
Pereslavl-Zalessky, Russian Federation
{asmirnov_1991, mannaz2012}@mail.ru, igor.p.tishchenko@gmail.com

***Abstract*—The article reviews the application of the path planning algorithm A* to meet the challenges of navigation of Mobile Technical Unit (MTU). In particular, the algorithm is applied to calculate the trajectory of bypassing obstacles. An algorithm for generating a map of location of nearby objects using stereo vision has also been developed, which can be used to determine their relative coordinates. As a result, a path of movement of the MTU and a method of bypassing obstacles in its path has been received.**

I. Introduction

Finding a way around obstacles is a typical problem in the task of search of path. There are algorithms which successfully cope with the task. Such algorithms include algorithms of path finding in a graph, such as breadth-first search [1] and depth-first search [2], as well as Dijkstra's [3] algorithm and best-first search [4] algorithm. All of the above algorithms provide adequate results, however, A* (A star) algoritm is the best one to find optimal ways in different spaces. This heuristic search sorts all nodes according to the approximation to the best route going through this node. Thus, A* takes the path length into account according to Dijkstra's algorithm and uses the heuristics of the of best-first search algorithm.

In practice, to use algorithms for constructing the path you need to have an idea of relative positions of obstacles and objects close to the robot. To determine the relative position of objects one can use the data analysis of the depth of the scene. These data may be provided by specialized scanning devices such as a scanning laser rangefinder (SLD) or lidar [5]. Also, data on the depth of the scene can be obtained using a stereo pair of cameras. With the help of a stereo pair of cameras one can generate a depth map and calculate relative position of objects. Use of stereo cameras is preferable, because of higher cost of lazer scanners.

Thus, having a stereo pair of cameras one can determine the location of obstacles and create a map of the scene, which can be used for pathfinding algorithm A *. To verify the algorithms the basic construction of the MTU was developed. The MTU is a mobile robot able to move in all four directions. Fig. 1 shows the basic construction of the MTU.

Basic construction of the MTU comprises:

1) Block of sensors. In this block there are various sensors of visible and invisible range, such as a stereo pair of cameras and rangefinders. It is also possible to install more complex sensors, such as lidar.
2) On-board computer. A single-board microcomputer Raspberry Pi or its analogs such as BananaPi and OrangePi can be used as a computer. To communicate with any external devices and manage them, Raspberry Pi has on-board interface called GPIO (General Purpose Input Output) [6]. It is a low-level IO direct control interface.
3) Block of engines. All hardware that allows MTU to move is in this block.
4) Cluster computing device. If necessary the MTU can transmit part of data for processing to the cluster computing device. Thus, resource-intensive data processing will take less time.

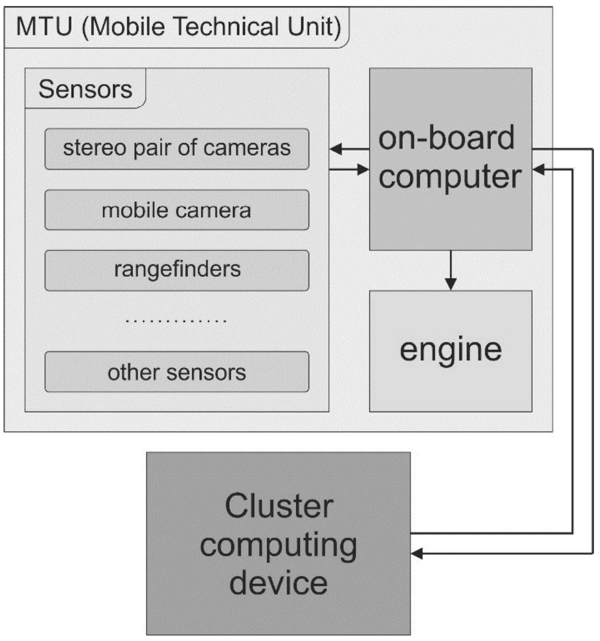

Fig.1. The basic construction of the MTU

II. Algorithm A*(A star)

To search for a route and build the motion path from the starting point to the ending point (from the point A to point B) algorithm A * was selected [7]. A* - in computer science and mathematics, is the algorithm Best-First coincidence search on graph, which finds the route with the Least Cost from one node (starting) to the other (destination, ending). The order of traversal of vertices is defined by a heuristic function that is usually referred to as F (x).

This algorithm was first described in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael. This was essentially extension of Dijkstra's algorithm created in 1959. The new algorithm achieved higher performance (by time) by using heuristics.

The principle of the algorithm:

1) 2 lists of vertices are created - Open list and Closed list. The starting point is added to the open list, the closed list is empty so far.

2) The path cost determined by the function F is calculated for each vertex. Also, each vertex stores a reference to the parent vertex from which it came.

3) From an open list of vertices the point with the lowest cost is selected. Let's call it X.

4) If X is the destination, it is assumed that the path is found.

5) Transferring X from the open list to the closed list.

6) For each of the vertices neighboring for X (let's denote this neighboring point as Y), the following is done:

a. If Y is already in the closed list of vertices, this vertex is skipped.
b. If Y is not yet in the open list, then the vertex is added to the list together with the reference to the parent vertex X and with the calculated value cost.
c. From the open list of vertices one with the least cost is selected. This vertex Y is transferred to the closed list of vertices and replaces the vertex X.
d. If the open list of vertices is empty, and the destination has not been achieved - then the route does not exist.

This algorithm has been studied and implemented in the language C ++. To display the map and draw the found route the funds of the library of computer vision OpenCV [8] were used.

The function of calculating the distance between two points was used as a function of determining the order of traversal of vertices F (x). This function is defined by the following formula:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

Where x_2 and y_2 are coordinates of the end point, and x_1 and y_1 are coordinates of the current point.

A scene map of the premises representing the binary image was created to test the algorithm. White color On the map designates walkable area, black color – obstacles and various objects on the path of the movement.

Below the result of the work of the algorithm is given. Fig. 2 shows the process of searching for the path (marked by line) from point A (the lower point) to point B (the upper point). Fig. 3 shows the found path (marked by a line). The path was obtained as a result of passing through the closed list of vertices, moving back from the final (destination) vertex (point B), passing from each point to its parent vertex until we reach the starting vertex (point A).

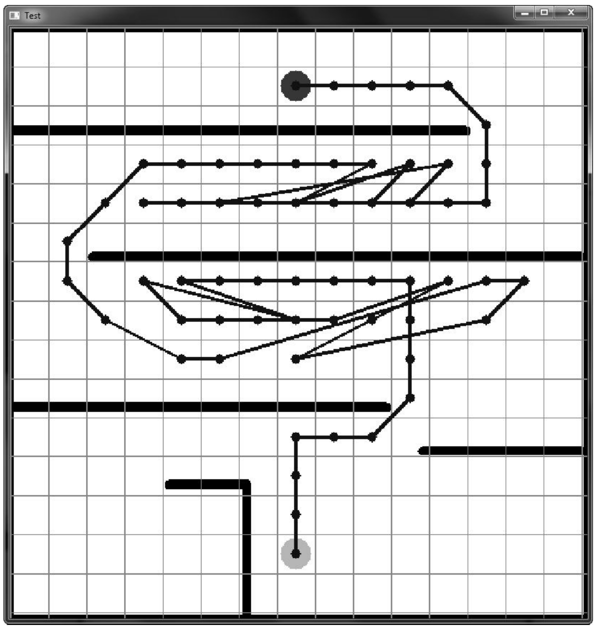

Fig. 2. The process of finding the path

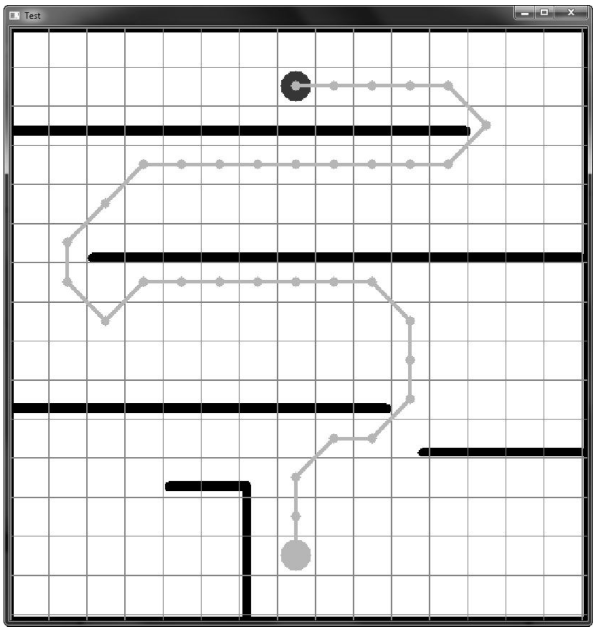

Fig. 3. Found path

During experiments it was found that the use of the function of calculating the distance between two points as a function of determining the order of traversal of vertices F (x) does not introduce any significant restrictions on the operation of the algorithm. Therefore, in conditions of the assigned task, it is not necessary to change the function of determining the order of traversal of vertices F (x).

III. THE ALGORITHM TO SIMPLIFY THE PATH (TRAJECTORY)

The simplification of the path (trajectory) is a procedure removing unnecessary vertices (points) from the list of vertices, which was received as a result of the algorithm A*. Unnecessary vertex is the vertex passage through which is not required when moving to the final vertex. The developed algorithm consists of the following steps:

1) The main vertex is selected. Initially, this is the very first vertex, which in this case is the starting point A.
2) A virtual segment is generated between the main vertex and each successive vertex from the list.
3) If you have managed to generate the virtual segment, then the next vertex is selected, and the main vertex remains the same until the virtual segment is over the obstacle.
4) If you have not managed to generate the virtual segment (the segment is over an obstacle), then the vertex which was before the current one, is selected as the main one. The vertex, which was the main one is preserved, and all the vertices between the preserved main vertex and the current main vertex are considered unnecessary. A segment is generated between the preserved main one and the current main one.
5) If you have managed to generate a virtual segment and the current vertex is the last one in the list then this vertex is the main one. In this case, the last vertex in the list is the point B. The path is simplified.
6) Segments are generated between the preserved vertices to display a simplified path.

After removing unnecessary vertices the angle between created segments is calculated. This angle is the angle of rotation of the MTU when moving in the found path. The angle is calculated by the formula:

$$\cos(\varphi) = \frac{(v_1, v_2)}{|v_1| \cdot |v_2|} = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \cdot \sqrt{x_2^2 + y_2^2}} \quad (2)$$

Where v_1 and v_2 - vectors made by segments the angle between which is calculated

The sign of the determinant of the matrix $\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$ determines the direction of rotation. It is the negative sign if you must rotate counter-clockwise and positive one when the rotation is clockwise. The result is shown in Fig 4.

IV. CREATE SCENE MAP FROM A STEREO PAIR OFIMAGES

For practical application of the above algorithms a map of relative location of nearby objects must be created. This map will be named map of scene.

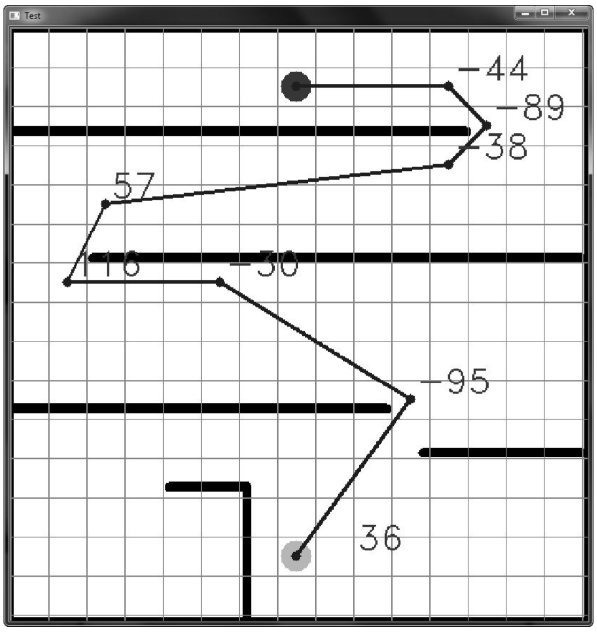

Fig. 4. Simple path and rotation angles

With a stereo pair of images it is possible to generate a disparity map, which can be used to create the map of scene. Stereo SGBM (Semi-Global Block-Matching Algorithm) [9-11] algorithm can be used to calculate the disparity map. This algorithm aims to minimize the following global energy function, E, for disparity image, D.

$$E(D) = \sum_p \Big(C(p, D_p) + \sum_{q \in N_p} P_1 I\big[|D_p - D_q| = 1\big] + \sum_{q \in N_p} P_2 I\big[|D_p - D_q| > 1\big]\Big) \quad (3)$$

With $P_2 \geq P_1$ where *E(D)* is the energy for disparity image, D *p*, *q* represent indices for pixels in the image, N_p is the neighborhood of the pixel p, $C(p, D_p)$ is the cost of pixel matching with disparity in Dp, P_1 is the penalty passed by the user for a change in disparity values of 1 between neighboring pixels, P_2 is the penalty passed by the user for a change in disparity values greater than 1 between neighboring pixels

I[.] is the function which returns 1 if the argument is true and 0 otherwise

The minimized function produces a perfect disparity map with smoothing governed by parameters P_1 and P_2; however, minimizing the function for a 2D image space is an NP-complete problem. The semi-global matching function approximates the 2D minimization by performing multiple 1D, or linear, minimizations. The matching function aggregates costs on multiple paths which converge on the pixel under examination. Cost is computed for the disparity range specified by the minimum disparity and number of disparities parameters. By default, the matching algorithm aggregates costs for 5 directions. You can set the full dynamic programming parameter to true to force the algorithm to aggregate costs for 8 directions.

Let, *S(p, d)* be the aggregate cost for pixel *p* and disparity *d*. Then

$$S(p,d) = \sum_r L_r(p,d) \tag{4}$$

Where *r* is a direction used for converging to the pixel p, $L_r(p,d)$ is the minimum cost of the path taken in direction r from pixel (p for disparity d)

The cost $L_r(p,d)$ is given in the following equation:

$$L_r(p,d) = C(p,d) + \min_i(L_r(p-r,d), L_r(p-r,d-1) + P_1, L_r(p-r,d+1) + P_1, \min L_r(p-r,i) + P_2) - \min_k L_r(p-r,k) \tag{5}$$

The equation uses the following costs to find the disparity by adding current cost, *C(p, d),* to previous pixel in direction *r*:

1) The minimum of the cost at previous pixel with disparity *d.*

2) The cost at previous pixel with disparity *d - 1* and *d + 1* with added penalty P_1.

3) The cost at previous pixel with disparities less than *d - 1* and greater than *d + 1* with added penalty P_2.

In order to limit the ever increasing value of $L_r(p,d)$ on the path, minimum value of the previous pixel is subtracted. The upper value of $L_r(p,d)$ is bounded by $C_{\max} + P_2$, where $C_{\max}$ is the maximum value of cost C. The cost function *C(p, d)* is computed in the following manner:

$$C(p,d) = \min(d(p, p-d, I_L, I_R), d(p-d, p, I_R, I_L)) \tag{6}$$

Where I_L and I_R are left and right rectified images, respectively

$$d(p, p-d, I_L, I_R) = \min_{p-d-0.5 \le p-d+0.5} |I_L(p) - I_L(q)| \tag{7}$$

The value of *C* is aggregated over a window of a user-defined size. After computing *S(p, d)* for each pixel *p* for each disparity *d*, the algorithm chooses the disparity which provides the minimum cost for that pixel.

The complexity of this algorithm is given in the following equation:

$$O(w \cdot h \cdot n) \tag{8}$$

Where *w* equals the width of the rectified image, *h* equals the height of the rectified image, *n* equals the number of disparities.

Stereo SGBM algorithm was implemented in the computer vision library OpenCV, with parameter list:

1) **minDisparity** (Minimum possible disparity value. Normally, it is zero but sometimes rectification algorithms can shift images, so this parameter needs to be adjusted accordingly).

2) **numDisparities** (Maximum disparity minus minimum disparity. The value is always greater than zero. In the current implementation, this parameter must be divisible by 16).

3) **SADWindowSize** (Matched block size. It must be an odd number >=1).

4) **disp12MaxDiff** (Maximum allowed difference (in integer pixel units) in the left-right disparity check. Set it to a non-positive value to disable the check).

5) **preFilterCap** (Truncation value for the prefiltered image pixels. The algorithm first computes x-derivative at each pixel and clips its value by [-preFilterCap, preFilterCap] interval. The result values are passed to the Birchfield-Tomasi pixel cost function).

6) **uniquenessRatio** (Margin in percentage by which the best (minimum) is computed, cost function value should "win" the second best value to consider the found match correct).

7) **speckleWindowSize** (Maximum size of smooth disparity regions taking into consideration their noise speckles and invalidate. Set it to 0 to disable speckle filtering. Otherwise, set it somewhere in the 50-200 range).

8) **speckleRange** (Maximum disparity variation within each connected component. If you do speckle filtering, set the parameter to a positive value, it will be implicitly multiplied by 16. Normally, 1 or 2 is good enough).

9) **P1** (The first parameter controlling the disparity smoothness).

10) **P2** (The second parameter controlling the disparity smoothness. The larger the values are, the smoother the disparity is. P1 is the penalty on the disparity change by plus or minus 1 between neighbor pixels. P2 is the penalty on the disparity change by more than 1 between neighbor pixels. The algorithm requires P2 > P1).

11) **fullDP** (Set it to true to run the full-scale two-pass dynamic programming algorithm. It will consume O(W*H*numDisparities) bytes, which is large for 640x480 stereo and huge for HD-size pictures. By default, it is set to false).

TABLE I. THE PARAMETERS AND VALUES FOR THE ALGORITHM SGBM

Parameter	Value
minDisparity	24
numDisparities	128
SADWindowSize	2
disp12MaxDiff	10
preFilterCap	4
uniquenessRatio	2
speckleWindowSize	155
speckleRange	1
P1	300
P2	2400
fullDP	true

This parameters is not ideal and may be changed.

Then two stereo pair of images were made with different number of objects (obstacles), for each of which a disparity map was generated. Fig. 5 - 10 shows the stereo pairs and the resulting disparity maps.

Further, to obtain data of depth scene it is necessary to apply the procedure for reconstructing 3D coordinates of the scene to the generated disparity maps and create a point cloud from the obtained coordinates. Translation of disparity map into the point cloud was made by the function reprojectImageTo3D [12]. This function is part of the computer vision library OpenCV. The result of function reprojectImageTo3D is a matrix containing coordinates X, Y and Z points (vertices). The matrix undergoes the process of filtration in which the ejections are removed first, that is the points whose coordinates (one or more) are equal to -∞ or ∞. The points where the Z coordinate is outside of certain range are also removed.

The algorithm of generating scene map from disparity map consists in projecting the coordinate Z, derived from a point cloud, on the axis OY. Analysis of the contours of found objects is applied to the resulting projection, whereby each found object is circled with a bounding box. Fig. 11 shows the disparity maps and projections on them.

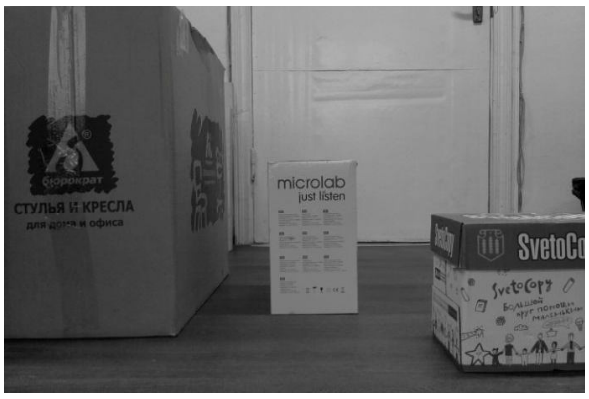

Fig.5. Stereo pair №1, Left frame

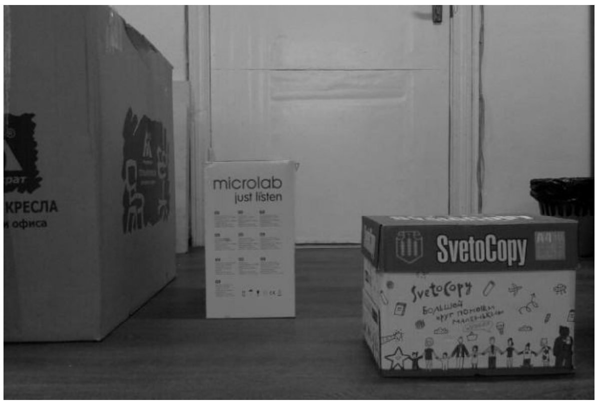

Fig.6. Stereo pair №1, Right frame

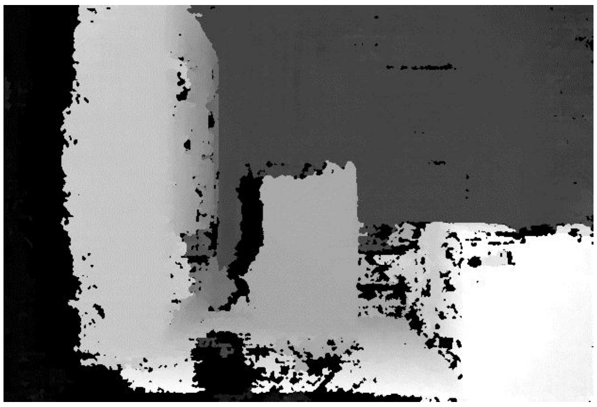

Fig.7. Stereo pair №1, disparity map

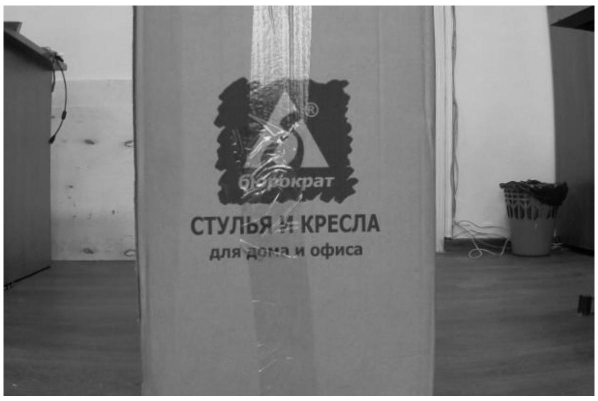

Fig.8. Stereo pair №2, Left frame

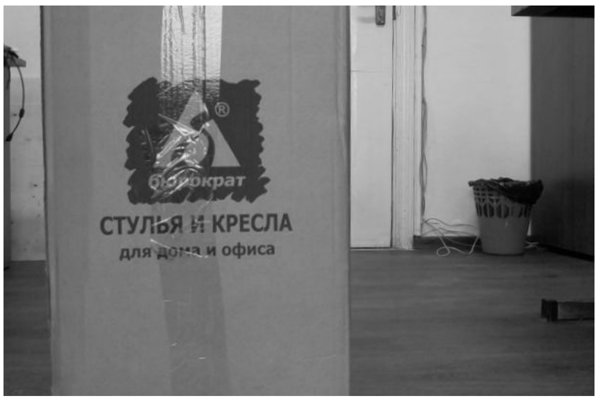

Fig.9. Stereo pair №2, Right frame

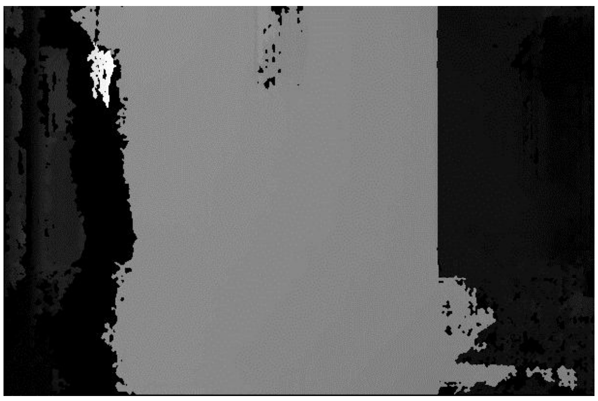

Fig.10. Stereo pair №2, disparity map

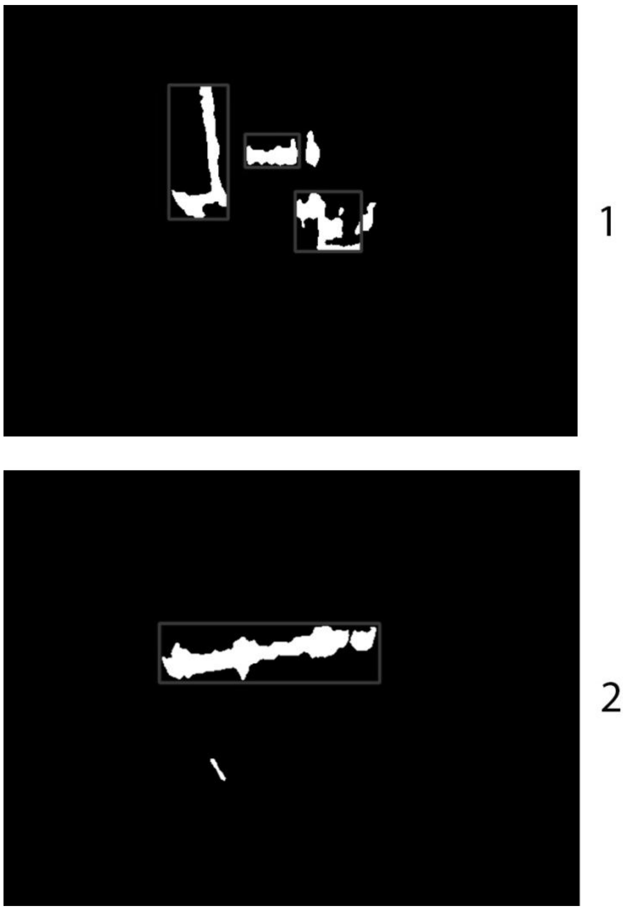

Fig. 11. The projection (1- The projection disparity map stereo pair №1, 2- The projection disparity map stereo pair №2)

Generated scene maps are shown in Fig. 12.

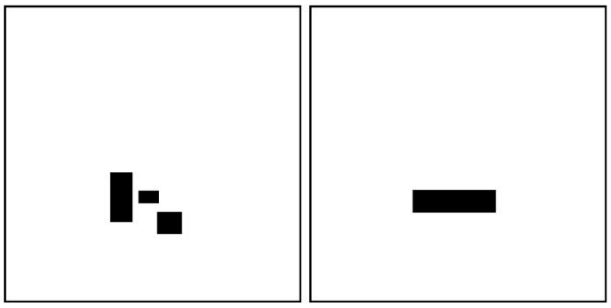

Fig. 12. Generated scene maps (left- №1, right - №2)

V. SOFTWARE IMPLEMENTATION

The concept of using the above algorithms with the MTU was created within the framework of the system developed in the Program Systems Institute of the Russian Academy of Sciences, which uses modular approach. Each independent algorithm was transformed into its module, as the system uses modular approach. Schematic representation of the modular approach is shown in Fig. 13.

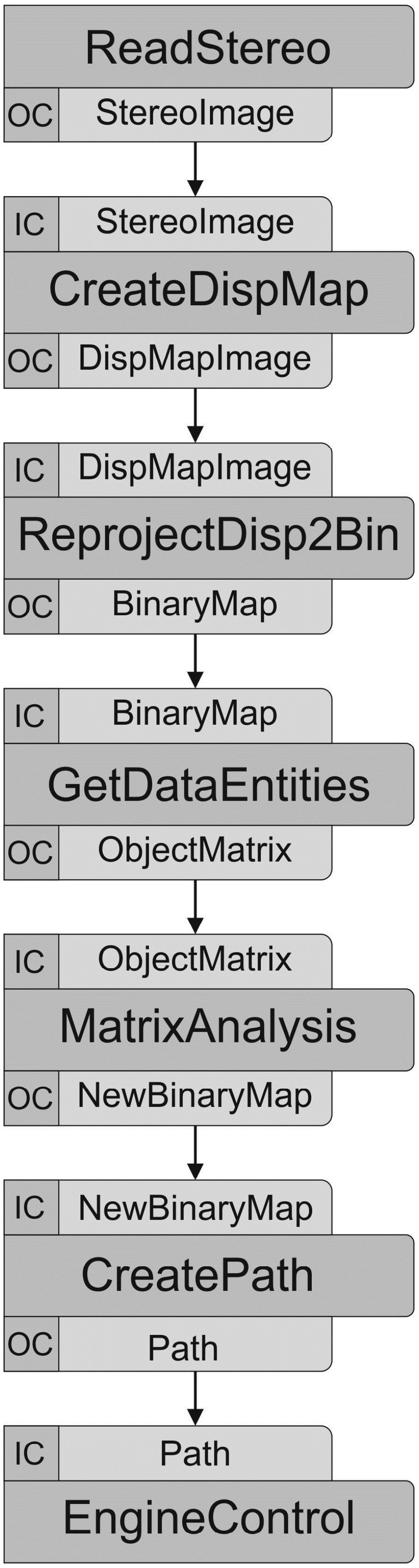

Fig.13. The list of modules.

The unit circuit is based on the principle "divide and rule", that is, each algorithm is a separate module. A module may have the input channel (IC), and the output channel (OC). Presented in Fig. 13 the circuit consists of the following modules:

1) ReadStereo Module. Reading and synchronization of data obtained from the stereo pair of cameras is taking place here. This module has only the output channel. Output channel: StereoImage – stereo pair of images.

2) CreateDispMap Module. The module creates disparity map. This module has an input channel and an output channel. Input channel: StereoImage – stereo pair of images; Output channel: DispMapImage – disparity map.

3) ReprojectDisp2Bin Module. The module performs recovery of 3D Coordinates by disparity map. Also the projected scene map (Fig. 11) is created in this module. The module has an input channel and an output channel. Input channel: DispMapImage – disparity map; Output channel: BinaryMap – binary scene map.

4) GetDataEntities Module. Here analysis of the map obtained from the previous stage is performed. In the module detection of all fixed objects is performed. This module has an input channel and an output channel. Input channel: BinaryMap – binary scene map (white color designates the object); Output channel: ObjectMatrix – matrix containing the coordinates of all found objects.

5) MatrixAnalysis Module. The module analyzes found objects and produces noise reduction. Input channel: ObjectMatrix – matrix containing the coordinates of all found objects; Output channel: NewBinaryMap – transformed scene map where all the objects that were selected are marked out with a bounding box (generated scene map).

6) CreatePath Module. Algorithm A * is applied to the resulting scene map in the module. This module has an input channel and an output channel. Input channel: NewBinaryMap – transformed scene map where all the objects that have been selected are marked out with a bounding box; Output channel: Path – List of reference points on the MTU path with indication of the angle of rotation.

7) EngineControl Module. The module is designed for direct control of MTU motors. This module has only the input channel. Input channel: Path – List of reference points on the MTU path with indication of the angle of rotation.

This circuit is able to operate on-board computer.

VI. Result

The path search algorithm A * was applied to the maps as well as the algorithm to simplify the path. The result is shown in Fig. 14 and 15.

As a result of application of the described algorithms to the scene map obtained using a stereo pair of cameras, the motion path of the MTU with obstacle avoidance was obtained. The path is comprised of the supporting vertices (points). Each vertex has an angle in degrees for rotation of the robot. When moving the robot must follow from one vertex to another, performing turns at the specified angle, until you reach the ultimate vertex.

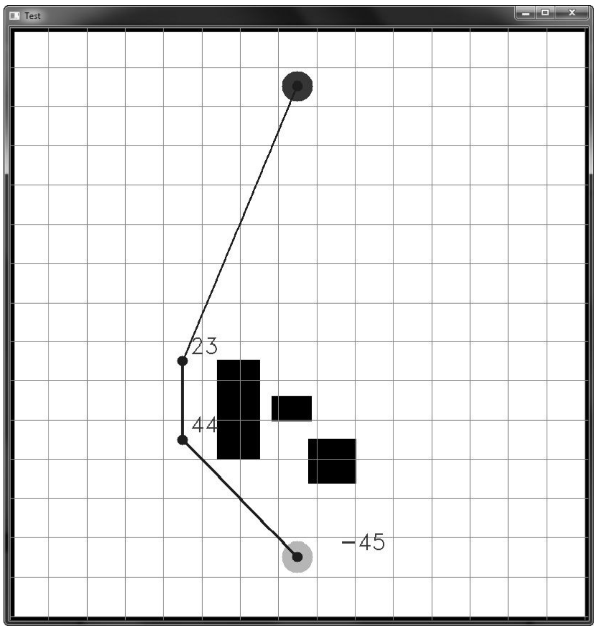

Fig. 14. The path and rotation angles for the map №1

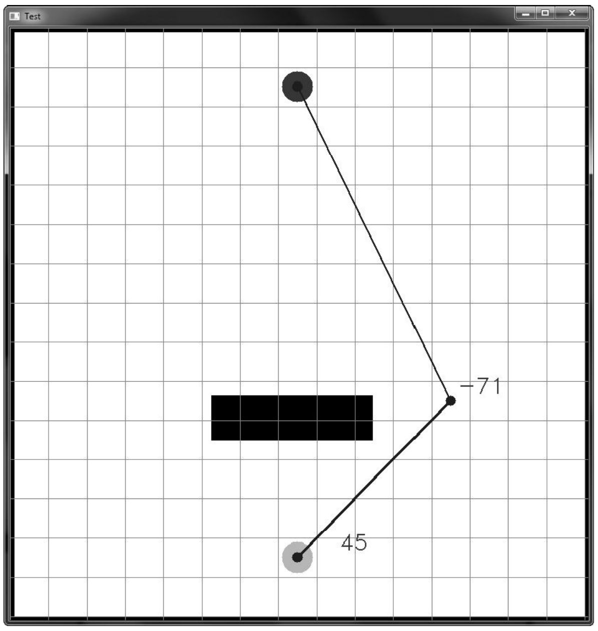

Fig. 15. The path and rotation angles for the map №2

Thus the path of the movement of the MTU has been obtained as well as the method of bypassing obstacles on its path with the use of stereo vision.

Acknowledgment

This work was supported by the Ministry of Education and Science of the Russian Federation, agreement № 14.607.21.0012 for a grant on "Conducting applied research for the development of intelligent technology and software systems, navigation and control of mobile technical equipment using machine vision techniques and high-performance distributed computing". Unique identifier: RFMEFI60714X0012.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms, The MIT Press, 2009.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Introduction to Algorithms, Third Edition, Clifford Stein 2005.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001.

[4] Russell S. J., Norvig P. *Artificial Intelligence: A Modern Approach.* — 2nd. — Upper Saddle River, New Jersey: Prentice Hall, 2003.

[5] Velodyne Lidar, Web: http://velodynelidar.com.

[6] Raspberrypi official website, GPIO, Web: https://www.raspberrypi.org/documentation/usage/gpio/.

[7] homepages.abdn, A* Pathfinding for Beginners, Web: http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm.

[8] OpenCV official website, OpenCV, Web: http://opencv.org/.

[9] OpenCV official website, StereoSGBM, Web: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#stereosgbm-stereosgbm

[10] Birchfield, S. and Tomasi, C. 1999. Depth Discontinuities by Pixel-to-Pixel StereoIJCV35(3):269-293.

[11] National instruments, Semi-Global Block-Matching Algorithm, Web: http://zone.ni.com/reference/en-XX/help/372916M-01/nivisionconceptsdita/guid-53310181-e4af-4093-bba1-f80b8c5da2f4/.

[12] OpenCV official website, reprojectImageTo3D, Web: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#reprojectimageto3d.