Path-building and localization system for mobile robotic platforms based on Raspberry Pi

E A Ivanov, A S Belyaev and O A Brylev

Tomsk Polytechnic University, 30, Lenin Avenue, Tomsk 634050, Russia

E-mail: eai13@tpu.ru

Abstract. The paper demonstrates a system for detection of location of robotic platform FESTO Robotino and optimal route building. It processes data from the camera and transmits control signals to the control system of the robot. The whole system is based on Raspberry Pi. It detects robot's current coordinates, current angular rotation, angular difference (difference between current and previous angular rotation) and displacement of the robot in its own coordinate system. It uses an ArUco marker, placed on the top of the mobile robot for that. System also builds an optimal path, when moving from one point of the surface to another, according to the permeability of the surface. The authors set the permeability of testing surfaces. Using that, a weighted graph is built through the centers of particular surfaces, which are detected via an algorithm on Raspberry Pi. The optimal path is constructed through the edges of the graph via modified Dijkstra algorithm.

1. Introduction

With technological and industrial development, modern mobile autonomous robotic systems gradually move from laboratories to the harsh environmental conditions. Predicting the environmental impact on the system, in this case, is usually impossible. It makes control systems for mobile robots become quite complicated. Among the basic mobile robotics' problems, the navigation task is still actual. It includes detecting current position of the robot via getting the data from the on-board sensor system, and path-planning, according to the permeability of the surface. The latter also consists of the optimal route building, obstacle-avoiding algorithm and others [1-5]. For this purpose, a system was constructed. It includes a testing surface [6-7], imitating the heterogeneity of the ground, a camera, placed right above that surface, and a single-board computer Raspberry Pi. The computer processes the image from the camera and the data from the robotic platform, and pushes signals to the mobile robot, moving on the stand. The structural scheme of the system and the image of the testing ground are presented in Figure 1.

The authors of [6-7] implemented a system for robot's position and angular rotation detection. Proposed system is also used for surface recognition, according to the data from the on-board sensor system. Specially created computer vision system, mounted above the the testing surface, is used for training and for evaluation of the results. It detects the location of the robot via the LEDs, set on it. However, such system is not sustainable, when there is no constant luminance. So, one of the objectives of current work is improvement of the location and angular rotation detection. The detection will be based on Raspberry Pi. It is enough for processing the image and all the algorithms. In addition, it is small enough for comfortable integration. The algorithm, detecting current location and angular rotation, includes linear correction of brightness in the picture, detection of the ArUco marker and conversion of coordinates of the ArUco marker into the robot's local coordinate system.

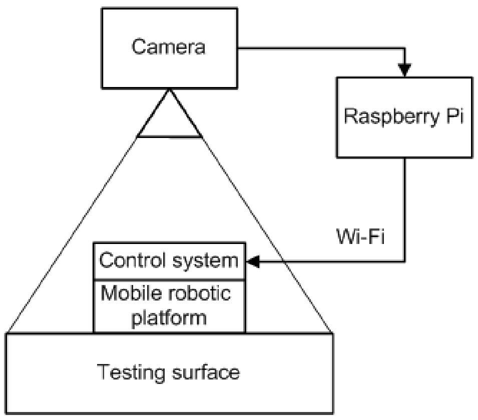

Figure 1. Structural scheme and image of testing surface.

2. Coordinates and angular rotation detection

In order to detect the location of the mobile robotic platform, a special ArUco marker was mounted on its upper side. A typical ArUco marker is demonstrated in Figure 2.

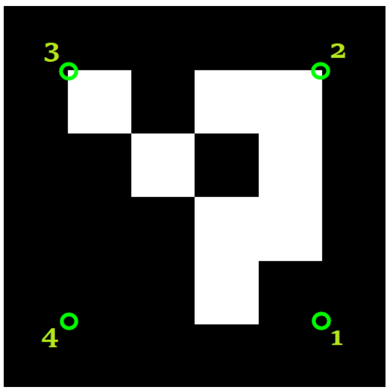

Figure 2. ArUco marker with detected corners highlighted with circles.

For ArUco marker detection a linear correction of the image was made. After that, a built-in function of OpenCV was executed for detecting four corners of ArUco marker. For proper robot controlling a set of data must be received: current location relative to the camera (in pixels), angular rotation relative to the camera, angular difference (difference between current and previous angle rotation), displacement relative to robot's local coordinate system (in millimeters). Having the coordinates of the ArUco corners, and using formula 1, we obtain the location of the center of the robot.

$$x = \frac{x_1 + x_3}{2}, \; y = \frac{y_1 + y_3}{2}, \tag{1}$$

where x_1, x_3, y_1, y_3 – coordinates of two diagonally oriented ArUco corners, x, y – ArUco marker center coordinates.

According to the orientation of the robot's local coordinate system, demonstrated in Figure 3, which was set as zero angular deviation, its evaluation is processed with formula 2.

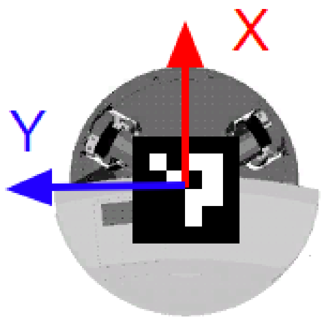

Figure 3. Coordinate system of FESTO Robotino.

$$\alpha = \begin{cases} \arccos \dfrac{x_2 - x_3}{\sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2}}, \alpha \in [0;\pi), \\ \arccos \dfrac{x_2 - x_3}{\sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2}} + \pi, \alpha \in [\pi;2\pi), \end{cases} \tag{2}$$

where x_2, x_3, y_2, y_3 – coordinates of two ArUco corners, forming the line segment, crossing the positive half-axis of OX.

Angular difference is obtained using formula 3.

$$\Delta\alpha = \alpha(t) - \alpha(t - \Delta t), \tag{3}$$

where $\alpha(t)$ – current angular rotation, Δt – sampling time.

Local displacement is obtained using formula 4.

$$\begin{pmatrix} \Delta x_{local} \\ \Delta y_{local} \end{pmatrix} = K \cdot \begin{pmatrix} -\sin\alpha & -\cos\alpha \\ -\cos\alpha & \sin\alpha \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \tag{4}$$

where $\Delta x, \Delta y$ – displacements of the robot relative to camera, $\Delta x_{local}, \Delta y_{local}$ – displacements of the robot in its local coordinate system, K – coefficient of conversion from pixels to millimeters.

In order to convert local displacement value from pixels to millimeters, the coefficient K is defined. It is calculated as the length of the side of ArUco marker in real, divided by its length in the image.

The above algorithm computes the location of the robot in global coordinate system, its angular rotation and displacement in local coordinate system.

3. Path-building algorithm implementation

Authors of [5-6] analyzed robot's movement across various types of surfaces. Based on that, the permeability of testing surface is evaluated. Each surface has its own amount of labor (in points of labor per pixel): Light-grey – 1, Pink and Light-green – 2, Green – 3, Beige, Blue and Dark-green – 4, Grey and Brown – 5, Red – 10.

Segmentation algorithm was implemented for detecting the centers of the surfaces. Firstly, the image is converted from RGB model to HSV model. Then, for each type of surface, threshold binarization is used. As a result, each surface is highlighted. A set of bit images for testing surface is represented in Figure 4.

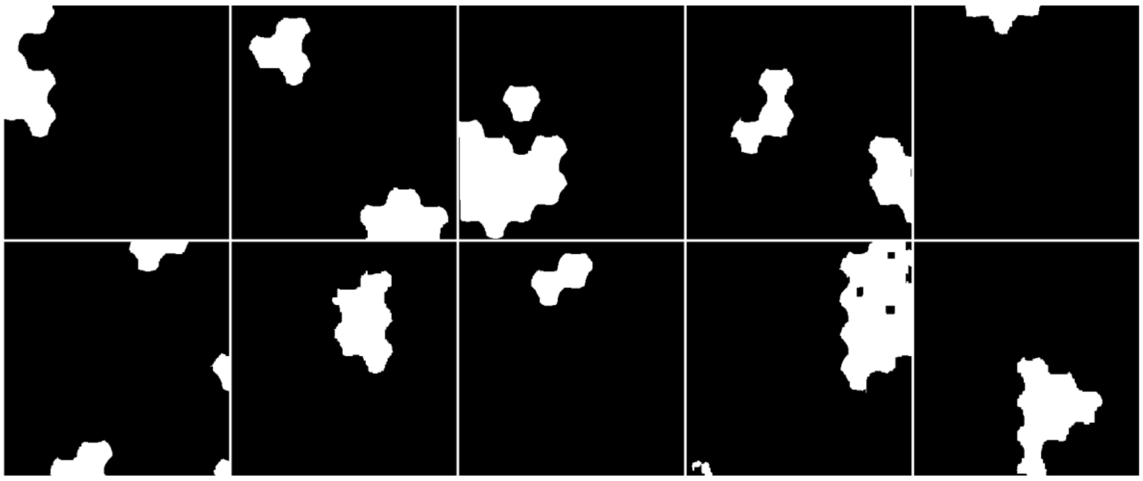

Figure 4. Set of bit images with highlighted surfaces.

Such algorithm is not among the most reliable, but, since the research is held in laboratory conditions, this method satisfies all the needs. However, it will be refined further. Next step is detection of the centers of separated surfaces. It is achieved via determination of mass center for each indispensable surface. Centers of surfaces are obtained using formula 5.

$$x_{c_i} = \frac{\sum_{j=1}^{n_i} x_{ij}}{n_i}, y_{c_i} = \frac{\sum_{j=1}^{n_i} y_{ij}}{n_i}, \tag{5}$$

where x_{c_i}, y_{c_i} – coordinates of the center of the *i*-th surface, n_i – amount of pixels of the *i*-th surface, x_{ij}, y_{ij} – coordinates of pixels of the *i*-th surface.

Then a weighted graph is constructed through the centers of each surface. In order to exclude the mutual influence of different types of surfaces, the limitations are introduced. All of the edges of the graph, except those, which cross only two surfaces, are removed. The final graph is demonstrated in Figure 5. This graph is weighed, according to the amount of points of labor of each surface. That is completed via counting the sum of point of labor for each pixel of particular edge of the graph.

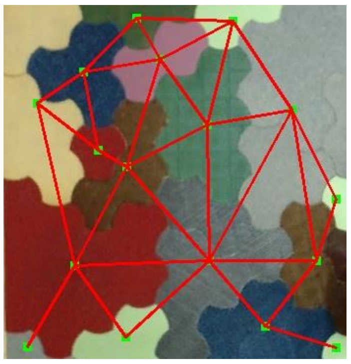

Figure 5. The final graph with limitations introduced.

Now, as the graph becomes weighted, the optimal path building algorithm is implemented. Dijkstra algorithm is used for that. It looks for the least cost route, when moving from one particular surface center to another. But this method defines only the amount of points of labor for the optimal path, but does not image the whole route. So, the algorithm was improved, so that it writes the sequence of the edges of the graph in the route. The whole Dijkstra algorithm and the supplement for it were implemented in C++. Its block diagram is presented in Figure 6.

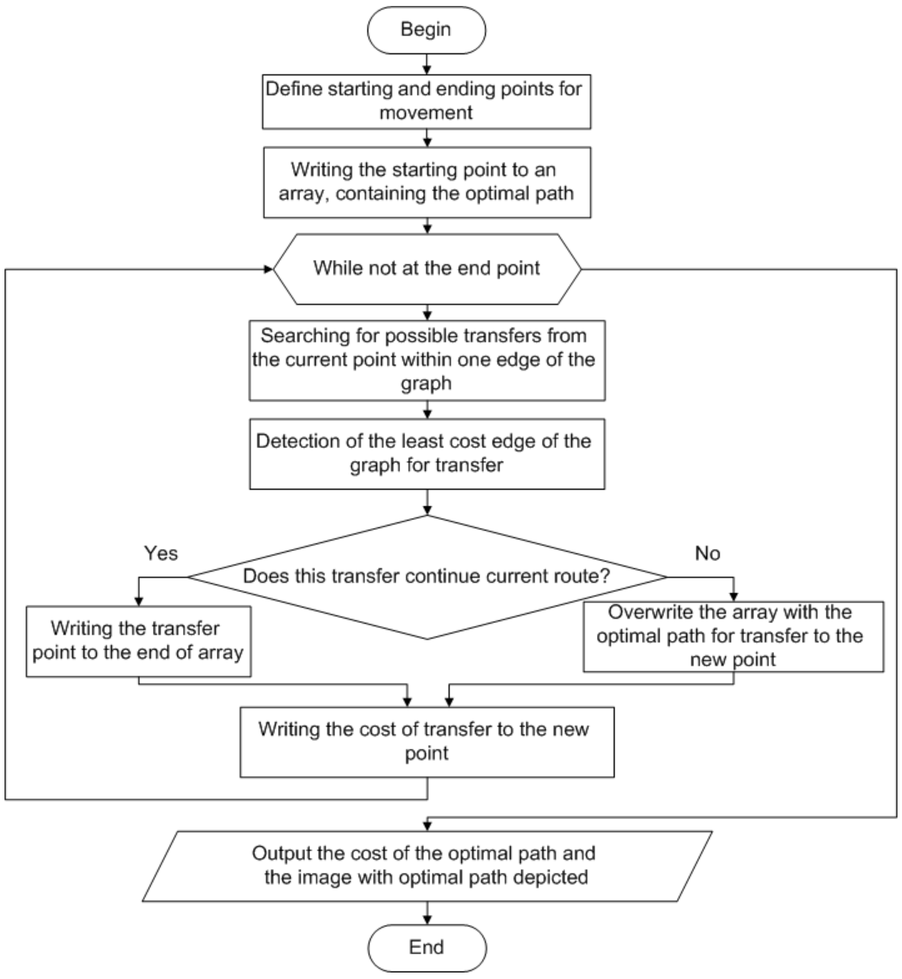
Begin
Define starting and ending points for movement
Writing the starting point to an array, containing the optimal path
While not at the end point
Searching for possible transfers from the current point within one edge of the graph
Detection of the least cost edge of the graph for transfer
Does this transfer continue current route?
Yes
No
Writing the transfer point to the end of array
Overwrite the array with the optimal path for transfer to the new point
Writing the cost of transfer to the new point
Output the cost of the optimal path and the image with optimal path depicted
End

Figure 6. Block-diagram of optimal path building algorithm.

As a result, the algorithm builds the optimal route. Examples of optimal routes are presented in Figure 7. Red routes are optimal, blue ones are the shortest (include the least possible pixels amount).

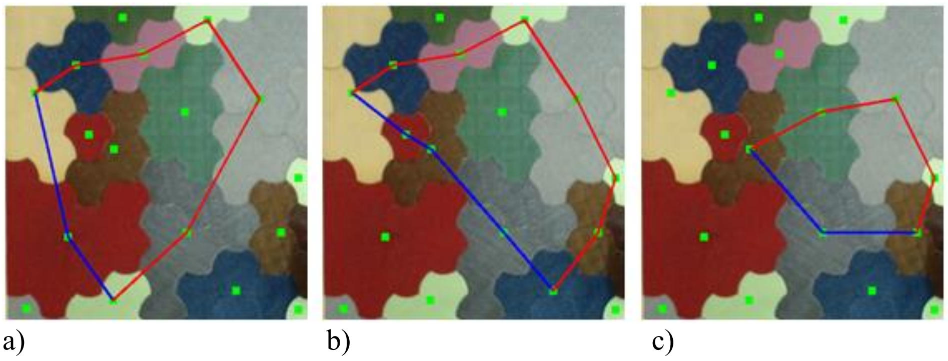

Figure 7. A set of examples of optimal (red) and shortest (blue) routes.

Figure 7a shows the optimal route with 1360 labor points and 617 pixels of length, while the shortest (blue) path takes 1634 labor points and 259 pixels of length.

In figure 7b, the optimal path takes 1308 points of labor and 578 pixels of length, and the shortest route is 1387 points of labor and 325 pixels of length.

Figure 7c represents the optimal route with 843 points of labor, 346 pixels of length, and the shortest route with 1015 points of labor, 236 pixels of length.

As it appears, the shortest path may not be the optimal one. That is an important aspect to be analyzed, as it influences the speed of robot's movement from one point to another.

4. Conclusion

As a result, this work provides an algorithm for coordinates and angular orientation detection, based on a computer vision system, based on Raspberry Pi. Optimal path building algorithm, which generates the least-labor route between two points, was also implemented. This whole system will further be used solving various navigation tasks and carrying out further investigations, related to mobile robotic platforms, moving across harsh surfaces.

References

[1] Howard A, Seraji H and Werger B 2002 Fuzzy terrain-based path planning for planetary rovers *IEEE Int. Conf. on Fuzzy Systems* pp 316-20

[2] Zhou L, Yang L and Tang H 2017 Research on path planning algorithm and its application based on terrain slope for slipping prediction in complex terrain environment *Int. Conf. on Security, Pattern Analysis, and Cybernetics, SPAC 2017* pp 224-7

[3] Iwasa A, Toda Y and Kubota N 2018 The return way path planning of an autonomous mobile robot considering traveling risk of the road *Proc. – Joint 10th Int. Conf. on Soft Computing and Intelligent Systems and 19th Int. Symp. on Anvanced Intelligent Systems, SCIS-ISIS 2018* pp 1406-9

[4] Guo Y, Song A, Bao J and Zhang H 2011 Optimal path planning in field based on traversability prediction for mobile robot *Int. Conf. on Electric Information and Control Engineering, ICEICE 2011 – Proc.*

[5] Tyryshkin A and Andrakhanov A 2009 Application of GMDH algorithms in the obstacle recognition problem for autonomous mobile robots *Pattern Recognit. Image Anal. 19 2009* pp 197-203

[6] Andrakhnov A and Belyaev A 2017 Navigation learning system for mobile robot in heterogeneous environment: Inductive modeling approach *Proc. of the 12th Int. Scientific and Technical Conf. on Computer Sciences and Information Technologies, CSIT 2017* pp 543-8

[7] Andrakhnov A and Belyaev A 2018 GMDH Based Learning System for Mobile Robot Navigation in Heterogeneous Environment *Advances in Intelligent Systems and Computing* vol 689 ed N Shakhovska, V Stepashko (Cham, Switzerland: Springer) pp 1-20