Autonomous Navigation System for a Wall-painting Robot based on Map Corners

João Carlos Virgolino Soares[1], Gabriel Fischer Abati[1], Gustavo Henrique Duarte Lima[2] and Marco Antonio Meggiolaro[1]

***Abstract*— Wall-painting is an important task in construction that has an increasing necessity for automation. This paper presents a navigation system for a wall-painting robot based on map corners. Using an occupancy grid map as input of the system, the proposed method is able to generate a path following the walls of the environment. The main advantage of the proposed methodology is the ability to work in non-convex and open maps. The system is implemented as a ROS package and is integrated with the ROS navigation stack. The method is tested in a simulator developed using ROS and Gazebo, a virtual environment that simulates the physics and dynamics of robotic systems.**

I. INTRODUCTION

Autonomous mobile robots are becoming each day more essential in different scenarios, such as industrial environments, hospitals, and offices. Wall-painting, for instance, has a high demand for automation. In general, painting work is a handmade process, and it is physically demanding to the workers, due to repetitive actions and chemicals in the paint that can cause health problems. Besides overcoming these issues, a wall-painting robot can also optimize the use of material and decrease the total work time.

A mecanum-wheeled robot [1] has high maneuverability for being holonomic, i.e., the robot is able to move in any direction in the plane while changing its rotation. Therefore, is highly suitable for the wall-painting task.

There are several wall-painting robots in the literature. However, their majority either follow a pre-designed path, which imposes a necessity to structure the environment and does not account for unwanted obstacles, or use a wall-following algorithm that cannot deal with complex maps. We propose a navigation system for autonomous wall-paining, designed for mecanum-wheeled robots, that works in unstructured environments and deals with non-convex and open maps.

The ROS Navigation Stack (RNS) [2] is a powerful tool that allows robot navigation with reliability, using an Adaptive Monte Carlo Localization (AMCL) system, a global planner, and a local planner.

[1]J. C. V. Soares, G. F. Abati and M. Meggiolaro are with the Department of Mechanical Engineering, Pontifícia Universidade Católica do Rio de Janeiro, R. Marquês de São Vicente 225, Gávea, Rio de Janeiro, RJ - Brazil virgolinosoares@gmail.com / fischerabati@gmail.com / meggi@puc-rio.br

[2]G. H. D. Lima is with the Department of Naval Engineering, Universidade Federal do Rio de Janeiro, Av. Pedro Calmon, 550, Cidade Universitária, Rio de Janeiro, RJ - Brazil ghduarte@poli.ufrj.br

978-0-7381-1153-7/20/$31.00

The RNS needs a desired goal as input. We propose an automatic goal generator based on the occupancy grid map of the environment. It allows the robot to follow a path through every corner, therefore going along the walls. Efficient computer vision techniques are used to detect the map corners and generate the goals with an offset, in order for the robot to maintain a distance to the wall. The proposed methodology is tested with Gazebo, an open-source 3D robotics simulation tool with a robust physics engine.

This paper is organized as follows. Section II presents related work, Section III details the proposed methodology, Section IV show simulated results and Section V presents the conclusions and propositions for future work.

II. RELATED WORK

There are several wall-painting robots proposed in the literature. Madhira et al. [3] developed a 2-DOF fixed robot for wall painting. Teoh et al. [4] developed a wall-painting robot supported by cables. The user defined a wall area to be painted with an interface, and the robot remained in a fixed place by two hanging points. The operation area was limited by the cables length.

Sorour [5] developed a differential drive painting robot with ten sonars sensors, being six for obstacle avoidance, and four for navigation. Despite having an obstacle avoidance system, it could not operate in non-convex maps. Megalingam et al. [6] proposed a mecanum robot with ultrasonic sensors to keep a constant distance from the wall. However, its wall-following navigation method could not deal with wall discontinuities, i.e., open maps.

Wall-following is a well-known navigation method [7], with several advantages for autonomous wall-painting. It does not need an operator, does not require construction site adaptation, and has a low computational cost.

There are several methods to perform wall-following. Wei et al. [8] proposed a wall-following algorithm based on virtual walls, using a differential drive robot with a LiDAR laser scanner. They created virtual walls using laser information to simplify inner and outer corners. As a result, this simplification increased the average speed and still guaranteed obstacle avoidance. However, without a robust navigation system, the robot was susceptible to lose track due to obstacles and walls discontinuities.

Lo et al. [9] developed an indoor surveillance robot with a wall-following method based on a fuzzy logic controller. The controller uses range sensors and a set of fuzzy laws. With this method, the robot was able to navigate through an

indoor environment. However, without a global localization system, the robot would struggle to return to its course in case of losing track of the wall.

Lee et al. [10] proposed a behavior-based fuzzy controller, composed of three sub-fuzzy controllers, each one referring to the directions of the robot. Such controller could get the robot through convex and concave maps. However, this algorithm cannot deal with unexpected obstacles that can cause the robot to lose the wall reference. Huang [11] developed a wall-following method using Lyapunov-based control with a corner detection algorithm, based on geometric relations of infrared sensor readings. The method was only tested on closed and convex maps.

We propose a method that can work autonomously in complex environments such as open non-convex maps, and deal with unexpected obstacles during the painting process.

III. METHODOLOGY

This work is performed under three main assumptions: (i) the map is given to the robot *a priori*; (ii) the map walls are perpendicular, and (iii) the robot has four mecanum wheels, a range-based sensor, and wheel encoders.

Figure 1 shows the flowchart of the proposed methodology. First, a goal generator algorithm receives the occupancy grid map and generates the goals that are sent to the navigation system, which also receives sensor information and localization estimation. Finally, the navigation system sends movement commands to the base controller. The object detection module constantly evaluates if there is an unmapped object in the path of the robot.

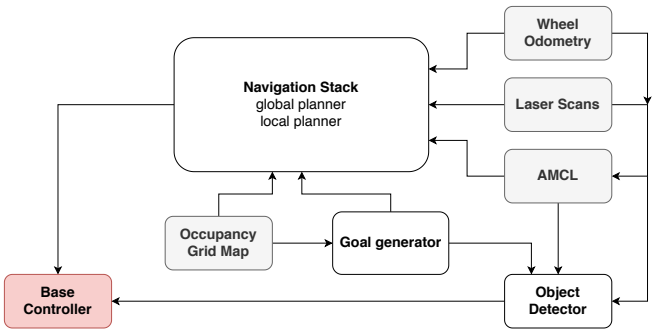


Fig. 1: Flowchart of the proposed methodology

A. Occupancy Grid Map

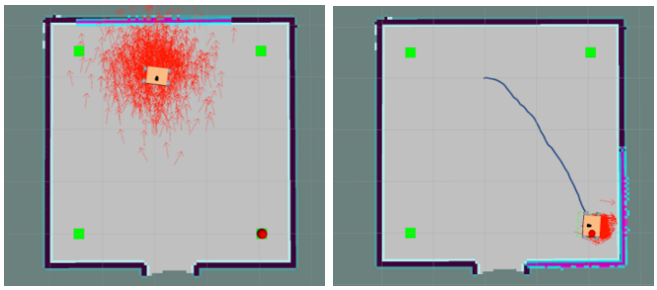
Occupancy grid map is a discrete representation of the environment by an evenly spaced grid. Each cell represents the presence or absence of an obstacle. The white color represents an empty space, black represents an occupied space, and gray represents unknown information. The grid map is created by a robot equipped with a laser scanner and odometry, performing Simultaneous Localization and Mapping (SLAM). Fig. 3a shows an example of an occupancy grid map used in this work, created using the ROS package Gmapping.

B. ROS Navigation Stack

The ROS navigation stack is a robust navigation system that receives as input odometry, range measurements, the initial pose of the robot, and the desired goal, and outputs velocity commands to the mobile base.

It uses the AMCL algorithm [12] to localize the robot in the map, using as input the given initial pose, and the sensor information. The output is the estimated pose of the robot. AMCL uses a set of weighted particles to represent the pose belief, which are updated when the robot receives more information about the environment.

Figure 2a shows the initial estimated pose of the robot together with the particles represented by red arrows. The particles are spread due to the high uncertainty about the pose. After several measurements, the system is more certain about the pose and the particles are more concentrated, as shown in Fig. 2b.



(a) AMCL - Initial estimation (b) AMCL - After resampling

Fig. 2: Robot localization

The RNS is composed of a global and a local planner. The global planner analyses the map and the goal in order to calculate the optimal trajectory, while the local planner transforms the global trajectory into local strategies considering the robot constraints.

C. Map Erosion

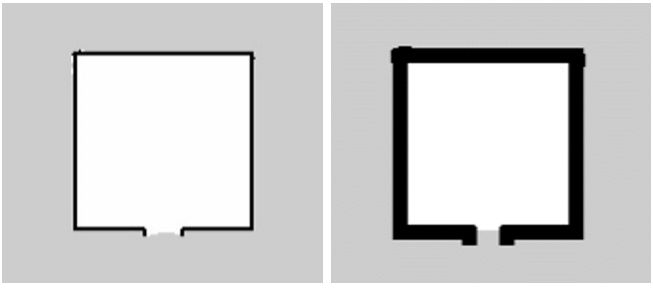
Erosion is a basic operator in mathematical morphology. A kernel is used to keep or discard pixels in the image, decreasing the size of the interior, and consequently, increasing the thickness of the edges [13].

We use the erosion operation to create an offset of the map edges, as shown in Figs. 3a and 3b. Figure 3a is the original occupancy grid map generate by the SLAM process, and Fig. 3b is the same map after the erosion. Using this process, the goal generator is able to create goals with a distance to the walls.

D. Harris Corner Detector

The Harris Corner Detector was proposed by Harris and Stephens [14] in 1988. It is an efficient method to find corners, commonly used in computer vision.

Figure 4 shows the detected corners in the map of Fig. 3b. We use the detected corners as goals to be sent to the navigation stack. Using this method, the robot can go through every edge of the map, including openings, even if it is a non-convex map.



(a) Original Map (b) Eroded Map

Fig. 3: Difference between the original and eroded map

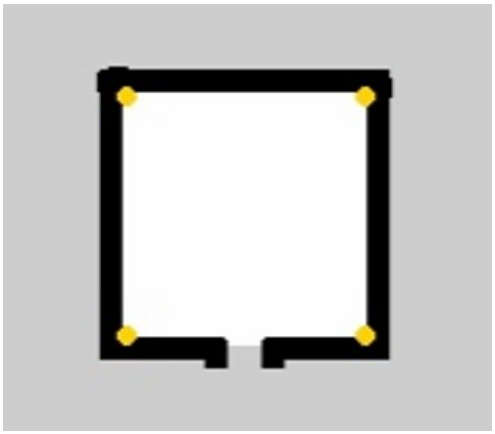


Fig. 4: Harris Corner Detection

If the map has an irregularity, some redundant points can be detected. We propose a method to delete such points, described in Algorithm 1.

Algorithm 1: Redundant point filter

Input: $Points$, $distThreshold$;
for $i = 0$ *to size(*$Points$*)* **do**
  $current\_point = Points[i]$;
  for $j = i + 1$ *to size(Points)* **do**
    $next\_point = Points[j]$;
    $dist =$
     $euclidean\_dist(current\_point, next\_point)$;
    if $dist < distThreshold$ **then**
      $eliminate\_point(Points[i])$;
    end
  end
end

E. Sorting goals

The Graham scan algorithm [15] is a method to find a convex hull from a given set of points. It first localizes the bottommost point as a reference. If there is more than one, it chooses the rightmost point. Once found the reference point, it is placed at the first position of the input list of points. The remaining points are sorted based on their locations with respect to the reference. After sorting, the graham scan filters the points that do not fit in the resulting convex polygon.

We propose a modified version of the Grahan scan, shown in Algorithm 2, to sort the navigation goals in order to the robot always follow the edges of the map counterclockwise.

Algorithm 2: Sort Goals - Modified Graham

Input: list of pixel coordinate points;
for p_i *in list* **do**
  find min($p_i.y$) and min($p_i.x$);
  $p_0 = p_i$;
end
swap p_0 with $list[0]$;
for *idx = 1 to size(list)* **do**
  $d_1 = list[idx + 1]$ distance to p_0;
  $d_2 = list[idx + 2]$ distance to p_0;
  $\theta_1 = list[idx + 1]$ angle in relation to p_0;
  $\theta_2 = list[idx + 2]$ angle in relation to p_0;
  if $\theta_1 < \theta_2$ **then**
    $list[idx + 1]$ comes before $list[idx + 2]$;
  end
  else if $\theta_1 = \theta_2$ **then**
    if $d_1 < d_2$ **then**
      $list[idx + 1]$ comes before $list[idx + 2]$;
    else
      $list[idx + 2]$ comes before $list[idx + 1]$;
    end
  end
  else
    $list[idx + 2]$ comes before $list[idx + 1]$;
  end
end

Figure 5 show the result of the sorting algorithm. The red numbers are the initial random corner order. The green numbers represent the sorted order.

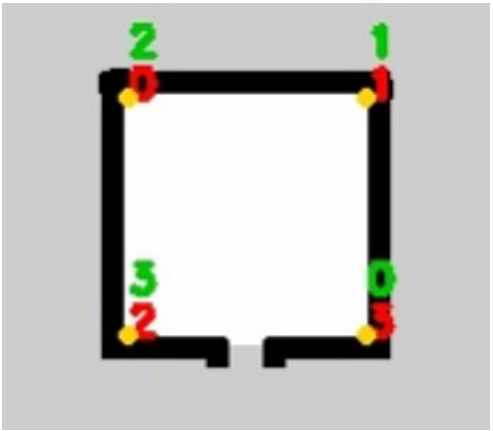


Fig. 5: Sorted goals

F. Goal generator

The goal generator module is described in Algorithm 3. It combines the techniques of Harris Corner Detection and

Map Erosion to find an offset of the map corners, in order to generate goal coordinates that will be send to the RNS. The goals are sorted in a counterclockwise manner using the modified Graham scan algorithm, and their coordinates are converted from pixel values to meters. After these pre-processing steps, the robot navigates to each goal maintaining its orientation towards the walls until it passes along every edge of the map.

In order to decide the next orientation, the system uses the coordinates of the current pose of the robot, the previous and the next goal, to analyze if the current corner is open or closed. Depending on the case the robot decides if it should turn 90 degrees clockwise or counterclockwise.

Algorithm 3: Goal generator algorithm

```
Input: map image file;
Initialize ROS navigation stack;
Get map corners with Harris Corner Detector;
Redundant point filter;
Sort map corners with modified Graham scan;
Convert pixel coordinates to meters;
while Goals do
    if goal = first goal then
        move to goal;
    else
        stop robot;
        evaluate next goal orientation;
        if wrong orientation then
            turn;
        end
        move to goal;
    end
end
```

G. Object Detection

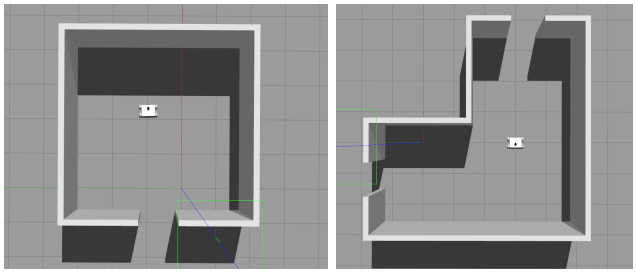
The object detection module receives the AMCL pose estimation, laser scans, and the next goal of the robot. It evaluates if the distance parallel to the movement of the robot, measured by the laser scanner, is lower than a threshold. If it is, then an unmapped object is in the path and the robot stops the movement.

H. Implementation Details

The proposed navigation system is implemented as a C++ ROS package. The OpenCV library is used for the Harris Corner Detection and morphological operations.

IV. RESULTS

The proposed methodology was tested using the ROS package Rviz, a 3D visualization tool, and Gazebo, to simulate the robot in the real world. Figures 6a and 6b shows the mecanum-wheeled robot in the two worlds of the gazebo environment used in this evaluation. All tests were performed on a laptop with an Intel Core i7 2.60 GHz and 16 GB of RAM running Ubuntu Linux 18.04 LTS.



(a) Rectangular World (b) L-shaped World

Fig. 6: Gazebo environment

Figure 7 shows the URDF (Unified Robot Description Format) model of the mecanum-wheeled robot used in the simulations. It has four independent mecanum wheels with odometry and a LiDAR sensor. The painting system is omitted.

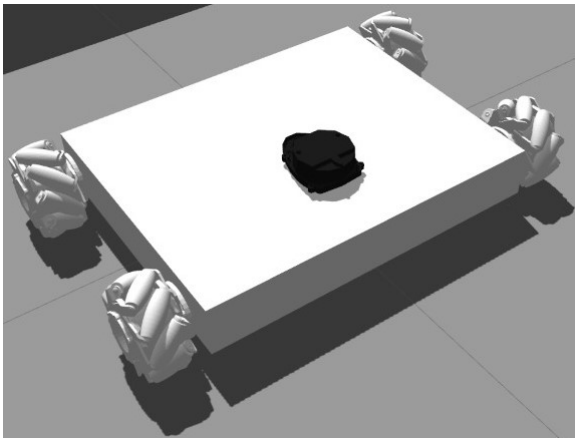
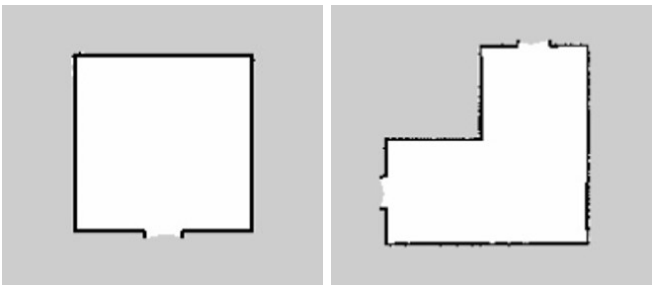


Fig. 7: Mecanum-wheeled robot used in the simulations

Figures 8a and 8b show the maps of the worlds shown in Figs. 6a and 6b. The rectangular map is open and convex, and the L-shaped is open and non-convex.



(a) Rectangular Map (b) L-shaped Map

Fig. 8: Test maps

A. Convex Map

First, the method was tested in the convex rectangular map. The robot starts from a generic pose in the map, goes to the bottom-right goal, faces the wall and proceeds until it returns to the first goal. Figure 9 shows the trajectory of the robot

in the rectangular map. The robot was able to complete the task.

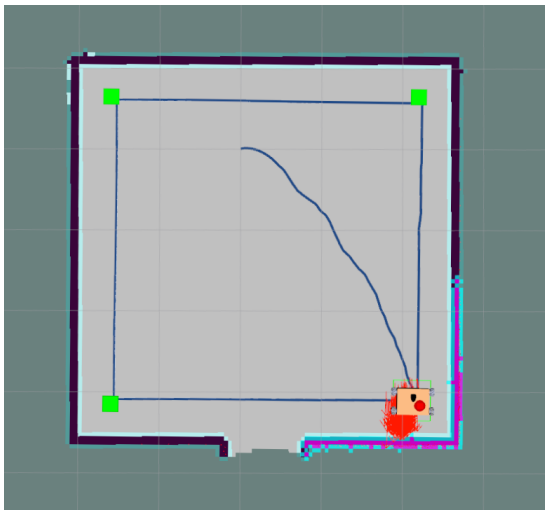
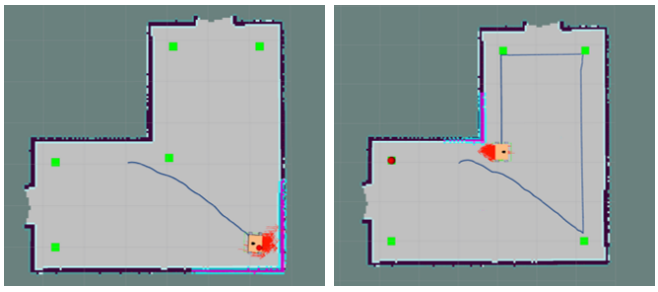


Fig. 9: Trajectory in rectangular map

B. Non-convex Map

The method was also tested in an open L-shaped non-convex map. Figure 10a shows the robot reaching the first goal after leaving the initial pose. Figure 10b shows the robot just after reaching the 4th goal. The red dot represents the next goal. It is noticeable that the robot rotates after every goal to always face the walls. Fig. 11 shows the complete trajectory. The robot was able to successfully complete the task.



(a) Reaching the first goal (b) Going to the 5th goal

Fig. 10: Trajectory in the L-shaped map

C. Unexpected Obstacle

To evaluate the effectiveness of our method against unexpected obstacles, we put a box in the path of the robot in the L-shaped map. After the robot starts moving, the object detection module discovers the presence of an unmapped object and stops the robot. Figure 12 shows the interrupted trajectory of the robot, and Fig. 13 shows the robot in the gazebo environment in front of the object.

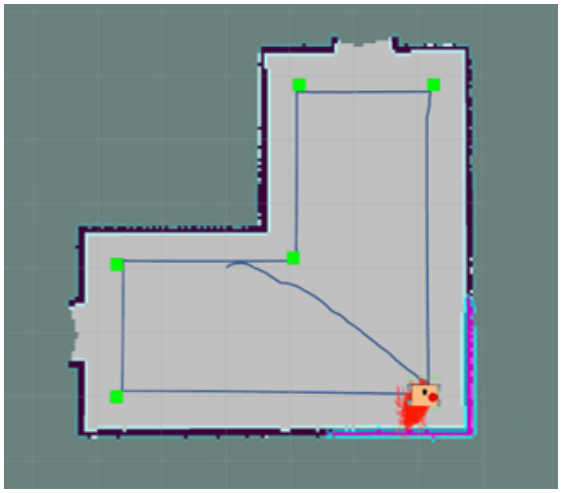


Fig. 11: Trajectory in the L-shaped map

Fig. 12: Interrupted task

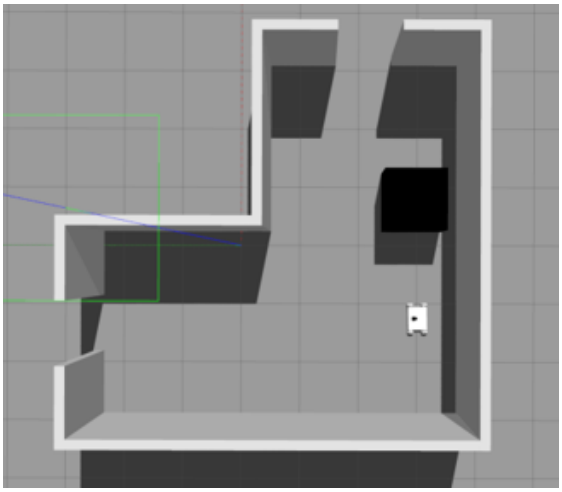


Fig. 13: Obstacle in gazebo

D. Numerical Evaluation

Table I shows the numerical results in both maps. The execution time corresponds to the time during motion, and the

processing time is the time to generate the goals. The robot achieved an average speed of 17.96 cm/s on the rectangular map, and 10.96 cm/s on the L-shaped map, both feasible results. The robot took longer on the L-shaped map due to the higher number of edges, as the maximum rotational velocity of the robot was set lower than the maximum translation velocity.

TABLE I: Numerical results

Map	Proc. time [s]	Exec. time [s]	Traj. length [m]
Rectangular	0.003	104.5	18.76
L-shaped	0.002	211.05	23.14

V. CONCLUSIONS

This work presented a navigation system for an autonomous wall-painting mecanum-wheeled robot. The system included a goal generator algorithm that made the robot travel along every edge of the map, and an object detection module that allowed the robot to deal with unexpected objects. We tested our system in open convex and non-convex maps, accomplishing the task in a feasible time in every tested scenario.

However, the system has drawbacks that need further improvements. For instance, the need for perpendicular walls. Despite being a reasonable assumption for the wall-painting application, it is a restriction that can limit the applicability of our system.

For future work, we aim to expand the methodology to maps without perpendicular edges, and improve the obstacle detection module to allow the robot to continue the process after the obstacle is removed. Also, we plan to compare different planners and parameters, and evaluate the optimal for our application. Furthermore, we intend to test our methodology in a real mecanum-wheeled robot, equipped with wheel odometry and laser scanner.

REFERENCES

[1] J.C.V. Soares, G. F. Abati, G. H. D. Lima, C. L. M. de Souza Junior, and M. A. Meggiolaro, Project and Development of a Mecanum-wheeled Robot for Autonomous Navigation Tasks. In Proceedings of the XVIII International Symposium on Dynamic Problems of Mechanics, 2019.

[2] Quigley, M., ROS: an open-source Robot Operating System. 2009 IEEE International Conference on Robotics and Automation.

[3] K. Madhira, S. Mehta, R. Bollineni and D. Kavathia, AGWallP: Automatic guided wall painting system, Nirma University International Conference on Engineering (NUiCONE), Ahmedabad, 2017, pp. 1-5.

[4] B. E. Teoh and S. V. Ragavan, "PAINTbot - FPGA based wall painting service robot prototype," 2011 IEEE Recent Advances in Intelligent Computational Systems, Trivandrum, Kerala, 2011, pp. 777-782, doi: 10.1109/RAICS.2011.6069415.

[5] Sorour, M., RoboPainter: a detailed robot design for interior wall painting. IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO), 2015.

[6] Megalingam, R. K., Darla, V. P., Nimmala, C. S. K., Autonomous Wall Painting Robot. International Conference for Emerging Technology (INCET), 2020.

[7] P. van Turennout, G. Honderd, and L. J. van Schelven, Wall-following control of a mobile robot, Proceedings 1992 IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 280-285 vol.1.

[8] X. Wei, E. Dong, C. Liu, G. Han and J. Yang, A wall-following algorithm based on dynamic virtual walls for mobile robots navigation, 2017 IEEE International Conference on Real-time Computing and Robotics (RCAR), Okinawa, 2017, pp. 46-51.

[9] C. Lo, K. Wu and J. Liu, Wall following and human detection for mobile robot surveillance in indoor environment, 2014 IEEE International Conference on Mechatronics and Automation, Tianjin, 2014, pp. 1696-1702.

[10] C. Lee, C. Lin, and H. Lin, Smart robot wall-following control using a sonar behavior-based fuzzy controller in unknown environments. Smart Science, 5:3, 160-166, 2017.

[11] L. Huang, Wall-following control of an infrared sensors guided wheeled mobile robot. IJISTA, 2009.

[12] D. Fox, KLD-sampling: adaptive particle filters. Advances in neural information processing systems, pp. 713-720, 2002.
Fox, D. (2002). KLD-sampling: Adaptive particle filters. In Advances in neural information processing systems (pp. 713-720).

[13] R. M. Haralick and L. G. Shapiro, Computer and robot vision, 1992.

[14] Harris, C. G., and Stephens, M. A combined corner and edge detector. In Alvey vision conference, 1988, Vol. 15, No. 50, pp. 10-5244.

[15] Graham, R.L., An efficient algorith for determining the convex hull of a finite planar set. Information Processing Letters, vol. 1, no 4, pp. 132-133, 1972.