VOL. 15, NO. 10, MAY 2020 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

PATH PLANNING USING THE A* ALGORITHM FOR THE MECABOT MODULAR SYSTEM

Ricardo A. Castillo, Wilhelm A. Miño and Juan R. Rodríguez
Engineering Faculty, Nueva Granada Military University, Cra 11# 101-80, Bogotá, Colombia
E-Mail: ricardo.castillo@unimilitar.edu.co

ABSTRACT

This paper describes the design and simulation (by software) of a system composed of a set of agent robots which will be coordinated so that they can form a collaborative multi-agent system oriented to the development of tasks of exploration, collection and transport of elements taking into account the environment of displacement. Each agent consists of the union of multiple MECABOT robotic modules, which were previously designed and built by the DAVINCI group of the Nueva Granada Military University. A strategy will be developed to coordinate agents (composed of modules) that will have a fuzzy logic component, which will give the system the ability to automatically make decisions in order to optimize system performance, reducing the time of operation and energy consumption . In order to increase the robustness of the system to disturbances in the operating environment and / or possible failures in agent components, a distributed command architecture will be developed under which each agent will have artificial intelligence algorithms in order to recognize said Events and adapt its operating plan autonomously. The validation of the operation of the heterogeneous robotic system will be done through the simulation of each agent and then the collective system using specialized software.

Keywords: auto-configurator, A star algorithm, modular robot, serpent robot.

1. INTRODUCTION

The term robot became popular with the success of the work R.U.R. (Robots Universales Rossum), written by Karel Čapek in 1920[1]. In the English translation of this work the Czech word robota, which means forced labor or worker, was translated into English as a robot. Depending on the environment in which the mobile robot is located, a specific type of movement mechanism is needed, be it wheels, legs, flying systems, swimmers or apodal, among others. Modular autonomous reconfigurable robotic systems or self-reconfigurable modular robots are autonomous kinematic machines with variable morphology. This kind of robot saves costs and time in the exploration of environments since a single robot explores a real place full of obstacles without the need of a human operator, thus canceling the problem of processing delays in teleoperation, as well as loss of time in the configuration of each module separately, problems that arise when these operations are performed by humans.

Modular robots are classified into 7 different types according to: the malleability of the material in which they were manufactured, whether they are flexible or rigid; the architecture, if they are robots in chain or lattice; the application scenario of the modular robot that can be for exploration, search and rescue; the means of displacement whether in water, earth or air; the control system, if it is centralized or distributed; and the joining mechanism that can be magnetic or mechanical. Next, the background of this type of robot is shown according to the classification mentioned above.

The following is the background of the architecture of modular robots of chain type[1], lattice[2], Hybrids [3] and mobiles [4]. Chain type robots are widely used due to the simplicity of their architecture. The most recent modular robots of this type are the MIT's ChaniForm with the ability to join up to 30 small modules[5], the Polibot G3[6], the MNS robot are its novel physical configuration of neural network [7]and the versatile Dtoo robot [8]. Lattice-type robots allow the configuration of modular robots as if they were the atoms of a body[9], for example the configuration of structures through several modules. The most recent modular robots of this type are MIT's M-Block which[10], although it does not have mobile external structures, is capable of self-configuration, the Mori is an innovative robot that combines modularity with origami robots in a single configuration [11]and the versatile Atron robot[12]. The hybrid type robots combine the strength in structure of the lattice type robots with the locomotion capacity of the mobile robots and the versatility of the chain type robot. The most recent systems of this type are the modular robotic system MRS [13] which has the advantages of lattice and chain type robots, the SMORES that has the three configurations of the modular robot architecture [13], the fast self-configuration robot M-Train III [14] and the MOSS [14] that makes modular robots commercially available. Modular mobile robots combine wheels and other types of mechanisms to improve the movement capacity of the unit module. A recent example of this type of robot is the AMOEBA-1 that additionally has the ability to tilt stability [15].

Modular robots are used for scenarios in which the configuration of environmental obstacles is not known, therefore they are used for rescue and exploration scenarios. Robots of this type are the ModRED [15] robot that, like this project was previously simulated in Webots, is designing for the extraterrestrial exploration as well as the NASA snakebot[16]. Most of the current modular robots are designed for the terrestrial environment as shown above, however, in the air environment the modular configuration can work as well as the terrestrial one. By joining several modules the modular drone becomes more robust and increases the degrees of freedom as more

modules join.[17] There are modules with the ability to configure themselves on the ground and take off from the air as the distributed flight matrix [18]and others that can configure themselves in the air as the modquad, which, for future developments, is expected to create dynamic bridges in abysses for the exploration of places with these characteristics.[19] Another means of displacement is water, currently there is a need to explore or perform long rescue searches in the marine environment, thinking about this need they designed the nautical modular robot. The environment requires supervision in order to control each of the variables that make a nautical [20]environment a healthy place for living beings, the modular robot was designed for this purpose, the eel robot can send the state of the lakes to scientists that,[21] according to the information provided, they can create strategies to improve the medium.[19] The mechanisms of union between modules are essential for the correct functioning stability of modular robots. There are two novel strategies, such as self-soldering connectors [22] and the E-Face connector[23].

Current version of the MECABOT module is smaller than its previous version but with greater capacity, when using metallic motors of 4 kg of torque in its pivots. This presents a more compact design by eliminating the rotational motors of the sides, leaving that space for the motors of the pivots, this adjustment does not affect the other electronic components in which a step-down regulator was included to protect the modules from current peaks Unlike the previous versions, the semimodules that make up the MECABOT are not the same, the rear semimodule was adapted to carry two batteries of 7 volts, 3 amps for powering the module components, giving greater autonomy to the no need a power source connected by cables and taking advantage of the wireless communication of the MECABOTS for remote control[24].

2. DEVELOPMENT OF THE MOTION SIMULATION ALGORITHM

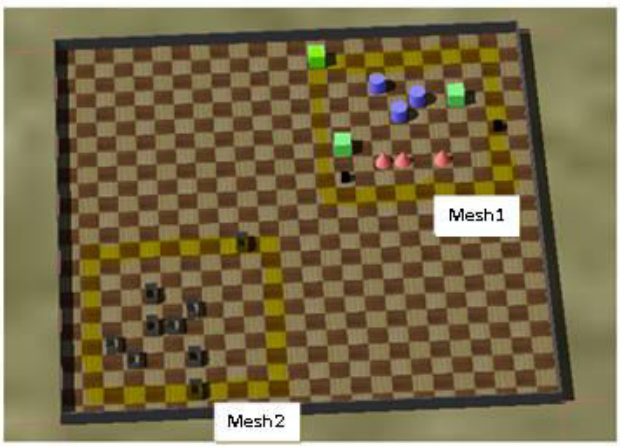


Figure-1. The yellow shading represents the two meshes that make up the world of webots.

The programmed world is divided into two meshes, one in which the MECABOTS are self-configured and another in which are the obstacles to which the MECABOT must adapt its morphology in order to reach the goal. It is done in this way to be able to solve the problem in different stages and thus facilitate its execution.

Next, the key points those allow to understand the program and avoid errors when programming in webots will be explained.

A. Supervisor definition

A supervisor is a special type of robot that is specially designed to control the simulation. A Supervisor has access to additional functions that are not available for a regular Robot. If a Supervisor contains devices, then the Supervisor's controller can use them. In the special case of this work, the supervisor fulfills the same task as an artificial vision system. It also emulates different random environments in program execution to show that the algorithm works correctly with any object distribution[25].

B. Creation of the cell object and its constructor

The goal of the algorithm is to take the MECABOT from the starting point entered by the user to the end point, for this the world must be divided into cells in order to solve the problem in stages.

To solve the problem in stages, an object called cell is defined which has as attributes: the coordinates in the grid of the initial cell and the parent cell, the estimated cost of the node from the initial cell to the goal (h), the cost of the path to this node (g) and the initial value of the heuristic function. The assigned initial values for these attributes are for the coordinates in 1 and the variables of the heuristic functions in 0.

The constructor of the cell object is created in order to assign different attributes depending on the coordinates and the value of the functions.

C. Assignment of start object, target object and obstacles

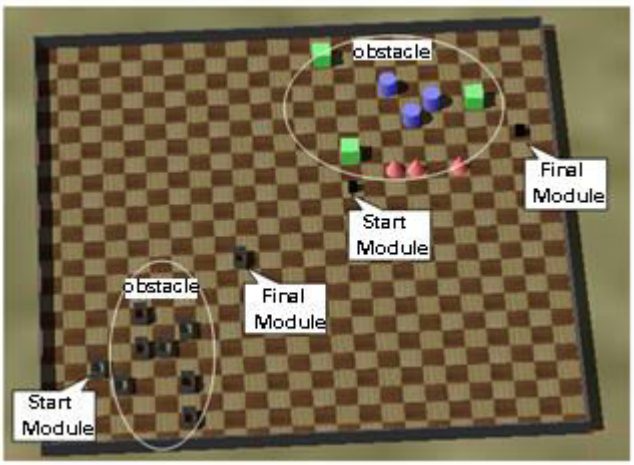


Figure-2. World is composed of 2 meshes.

The self-configuration of the MECABOTS is done one at a time, therefore, in order for each module to be able to join the corresponding one, the starting module, the final module and the obstacles in the world to reach the goal must be defined in each cycle (see Figure-1). This process is performed for each of the links in the chain that makes up the snake or the caterpillar.

By means of a for cycle, it is assigned, by means of a counter called number, the type of object of the MECABOT which can be the start, end object or obstacle. Initially the variable from type integer to string is transformed, this is done by the function str (). Then, the variable number is added with the variable fletter (Always with the value of B) to have a single string that accesses the DEF node of each MECABOT and thus automatically assign if it is an object of start, goal or obstacle.

Once the object is created, the node of the controller is accessed through the function getField ("controller") and the controller named MECABOTS is assigned to the start MECABOT by means of the setSFString () function and the empty MECABOT targets the controller, since it will not move, this is done by assigning it the value void.

After the start object has been created and the one designated to be the goal, obstacles are created by a for cycle in which it is evaluated with If conditions, if the number is equal to the start object or the target object, if not, the void value is assigned to the node of its controller, since the object must not be moved, this is done by the function setSFString (). The same procedure is performed for the other objects according to the amount specified by the user.

D. Emission of data from the supervisor to the MECABOT

The reception of data and its processing is done in the supervisor, since the equipment destined for supervision is more robust and fast. The final objective of the supervisor is to give the order of movement to the MECABOT, for this, the order represented by a character is sent, from: go to the right, to the left, continue in front and if the movement is wanted to be made to a high or low speed. Initially the variable is created for the sending node (wireless communication) by means of the code: sender = self.getEmitter ('emitter') and sends the processed command through the sender.send (order) function (see Figure-3).

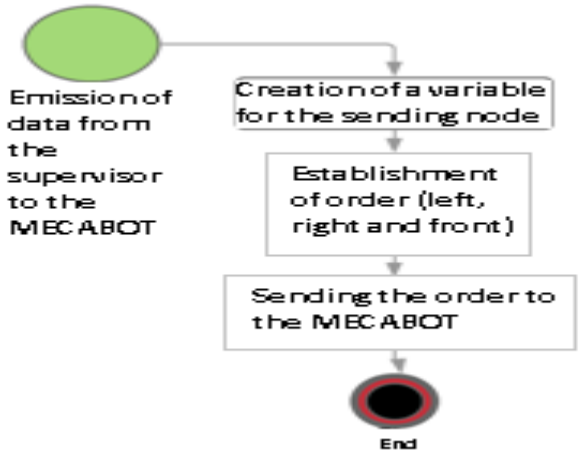


Figure-3. Emission of data from the supervisor to the MECABOT.

E. Initialization of discretized matrix of the work environment

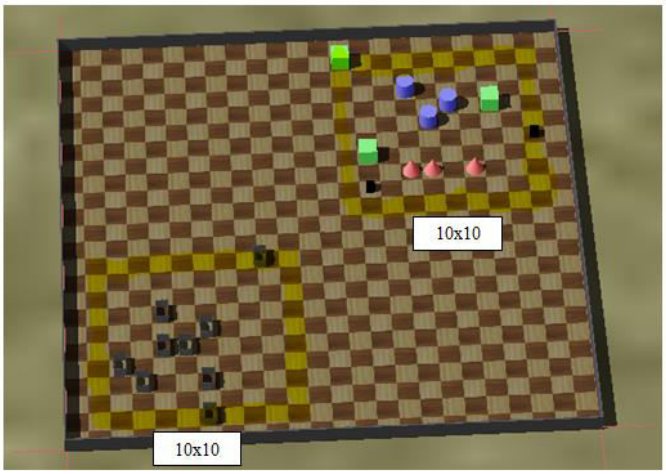


Figure-4. Dimensions of the world's meshes in webots.

For the creation of a matrix that represents the discretized work environment, the size of the world and the cells that make it up must be initially defined. For this case the size of the cell was defined as 0.24 for each side of the box that composes it, this selection was made based on the dimensions of the MECABOT and a little more space was added per cell to avoid collisions between the MECABOT and the obstacles (see Figure-5).

Once the size of the cell was defined, the size of the world was defined as 1.2, which corresponds to 5 times the value of the size of one side of the cell. Each cell corresponds to 4 cells in webots, this is defined by the variable num_celdas_por_lado = int ((tam_mundo / tam_celda) * 2) where the resulting value is 10 cells per side, meaning that the world is composed of 100 cells. Having already defined the dimensions of the world, we proceed to create a matrix of 10x10 zeros (see Figure-4).

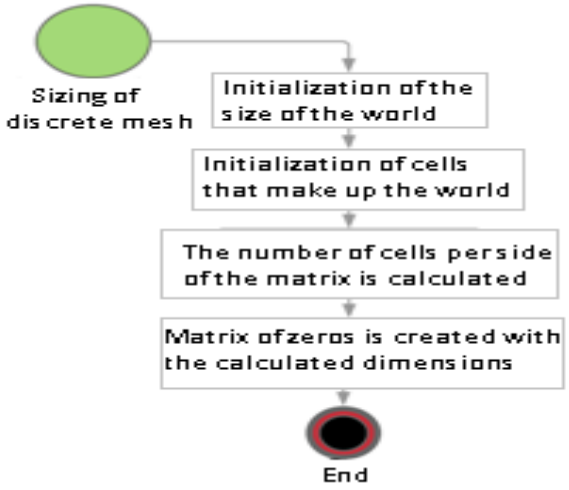


Figure-5. Dimensioning the discrete mesh.

F. Discretization of objects in the work environment

```
[[-1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, -1, 0, -1, 0, 0],
 [0, 0, 0, 0, -1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 4],
 [0, -1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, 0, -1, 0, 0, 0],
  [0, 2, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

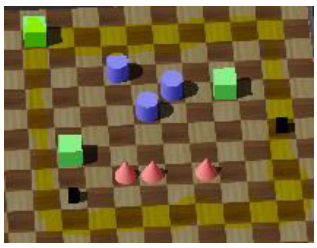


Figure-6. Discretized mesh vs. Real mesh in the simulation.

Initialized the matrix that represents the discretized work environment, we proceed to enter each of the obstacles that make up the world as well as the initial and final node of the trajectory. The node of the trajectory, defined in the previous steps as the initial, gets its position in x and y (by means of the function getField ('translation')) as well as its rotation with respect to the world in which it moves (by means of the function getField (' rotation ')).

By means of an if condition, the definition of the initial node is accessed, which, when the flag is equal to zero, defines the initial node of the chain of MECABOTS that overcomes various obstacles to reach the goal. When it is not equal to zero, the initial node of the MECABOT that defines a trajectory to form a chain of MECABOTS is defined.

Since these values are in non-discretized world terms, they are discretized by the function:

cell_ini_x = int (round ((50 * (robot_pos_x_ini) +27) /(6)))
cell_ini_y = int (round ((50 * (robot_pos_and_ini) +27) /(6)))

The same process mentioned is done for the definition of the final node and for each of the obstacles that the world contains (see Figure-7).

Defined the position in the discrete world of each of the obstacles, the initial node and the end, are entered into the matrix of the discretized world, using a for cycle, with a value of -1 for obstacles, 2 for the initial node and 4 for the final node, in Figure-6 the discretized right mesh can be observed on the left. The discretized mesh is automatically calculated by the supervisor.

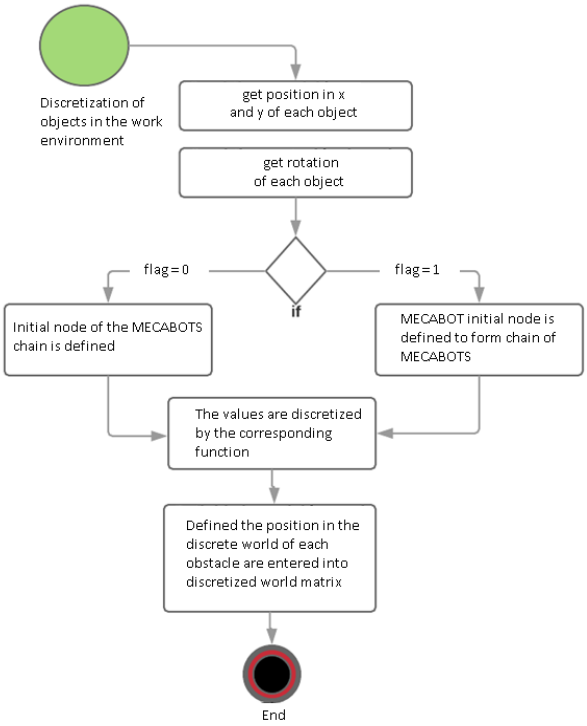


Figure-7. Discretization of objects in the work environment.

G. Execution function of the heuristic search algorithm A*

The entries of the function are the discretized world, the number of cells per face of the discretized world, the discretized cells of the obstacles in x and y of the initial node and the end node.

The trajectory begins empty, as well as the lists for the algorithm that are: OPEN, the cells without obstacles, CLOSED the cells with obstacles and COMES_FROM which is the cell of origin. Start by placing all the obstacles in the CLOSED list by entering the obstacles in the cells with the coordinates in x and y. The objects of the cell class are initialized for the coordinates in the cell matrix, the coordinates in the parent cell array, the estimated cost from this node to the target, the cost of the path to this node and the heuristic function. Then the starting point is placed in the OPEN list with the attributes of the cell previously named initialized[26].

In a while loop the path is searched, this cycle does not end until the cells in the OPEN list are finished, in this cycle the current node with the lowest value f (n) of the OPEN list is obtained, if the returned path is returned, the current node is removed from the OPEN list and placed on the CLOSED list. Then the current adjacent successor node is analyzed, if the adjacent node is in the CLOSED list, an iteration is skipped. Then, a tentative

value of the function g (n) is calculated, if the neighbor is not in OPEN or if gnthentic <g (neighbor), the values of the neighbor are updated and placed in the COMES_FROM list, if the neighboring value it is not in OPEN, it is added to OPEN. If there are no nodes in the OPEN list, there is no path that leads from the starting point to the goal. If they do exist, the trajectory is returned[26].

List of options required to correctly execute algorithm A *:

- The calculation of the distance between two cells, is done by the equation of distance between two points, which is equal to the sum of the squares of the difference between x and y, and the square root of the result.

- To verify if a node is within a list, it is evaluated by means of an if condition, if the current value is equal to one of the values in the list and it is passed through each position of the vector, by means of a for cycle.

- The return of the index with the lowest value in f (n), the input variables are the list, the size of the list, the position of the target cell in x and y.

The algorithm starts by reviewing all the nodes that are currently in the OPEN list. Then the value of f (n) of the current node is added to the array, if the target node is found, the index of the target node is saved and the flag is activated, otherwise, if the temporary array does not have a null size, the index of the node with the least value in the temporary array is calculated by means of the min () function. If the temporary array is null in size, it is empty, therefore there are no more available paths to evaluate for the search. The index is returned in the list corresponding to the minimum value.

- The function to find the neighbors, expands a node and returns the list of successors with their corresponding calculated values of f (n). Initially the neighbors located in the positions are proposed: up, down, left and right. Then, the successor values are assigned in x and y. If there is no object in the cell located in front of the current node, the neighbor vector is expanded to the diagonals to find the shortest path, otherwise the neighboring vector remains with the basic values mentioned initially to avoid collisions with the objects of the world in webots.

The 8 cells surrounding the current node are evaluated, as long as the node is within the limits of the array, and the neighbor is not an obstacle. There is the cost of traveling to the node g (n), the estimated distance between the node and the goal h (n) and with these two values, f (n) is calculated. The successor is created and added to the adjacent list and returned.

- The function to reconstruct the trajectory, starts an empty vector of the trajectory, the current node (at the beginning, the node of the goal) is added to the trajectory. While the current node is within the COMES_FROM list, the new current node will be the parent of the current node. The current node is added to the trajectory, if the parent is the starting point, it is added, the reconstruction is completed and the trajectory is returned.

3. TESTS AND RESULTS

Tests are performed on the programmed algorithm, the initial location of the MECABOT as well as that of the obstacles is random. Chains were tested from 2 modules to 10 connected in series. We evaluated the number of cells traveled by each MECABOT to reach the goal, the number of obstacles that had to evade to reach the goal, we measured the time it took for the MECABOT to calculate and also to travel the trajectory. Then the graph compares the behavior of the variables evaluated with respect to each other.

A. MECABOTS autoconfiguration tests varying the number of modules

A total of nine tests were performed on the algorithm, these tests were performed for different amounts of MECABOTS, from 1 to 10 modules. The objective of this MECABOTS autoconfiguration test is to demonstrate through simulation, the reliability of the algorithm to auto configure overcoming obstacles along the trajectory. In tests 1 to 9, the calculation time of the path and the execution time of the trajectory were used as performance indicators to evaluate the auto configuration with the input parameters.

As can be seen in Figure-8 as the number of cells covered by the MECABOT increases, the trend is that the execution time increases, however it can be observed, in some cases this relationship is not directly proportional and this must be to that in some trajectories the robot had to make more or less turns.

In the tests it can also be observed that the robot calculated the shortest path in all the trajectories carried out. In the Figure-9 it can be observed that the maximum time that the algorithm delays calculating the trajectory that the MECABOT must follow is of more or less 0.008 seconds, which allows that it can be seen how the trajectory is executed in real time and without delays important that damage the final purpose of the MECABOT that is self-configured to perform exploration activities.

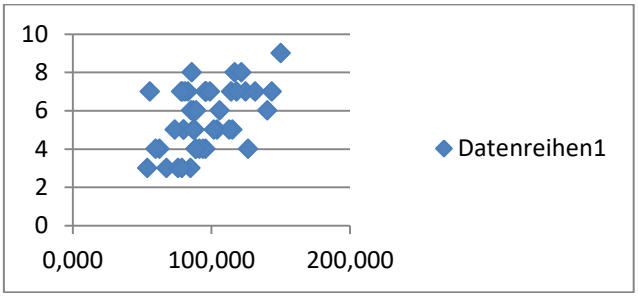


Figure-8. Path execution time VS number of cells traveled.

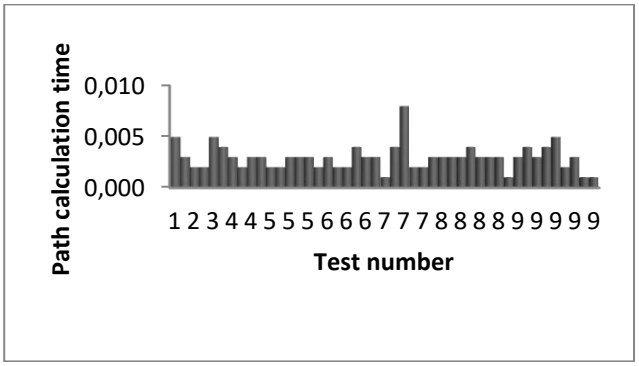


Figure-9. Path calculation time VS test number.

B. Test

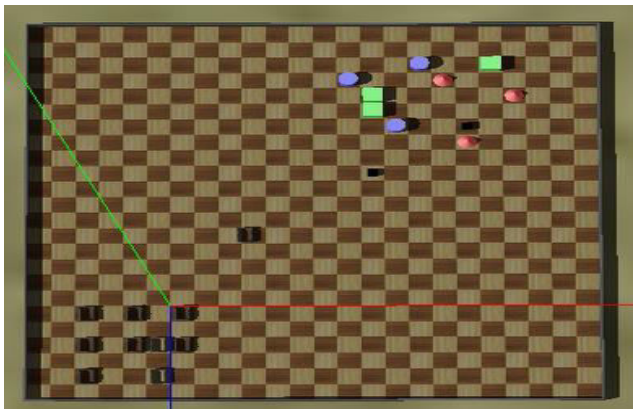
The objective of this MECABOTS auto configuration test is to demonstrate, by simulation, the reliability of the algorithm to auto configure, overcoming the obstacles along the trajectory with 9 MECABOT modules located randomly in space. The calculation time of the trajectory and the trajectory execution time to evaluate the auto configuration with the input parameters were used as performance indicators (See Table-1).

Table-1. Mesh data from MECABOTS test.

Chain module	Number of modules in total	Number of cells traveled	Number of obstacles overcome	Path calculation time (seconds)	Path execution time (seconds)
First	10	6	1	0.003	140.194
Second	10	7	4	0.004	98.882
Third	10	7	3	0.003	118.193
Forth	10	7	1	0.004	131.493
fifth	10	4	0	0.005	93.842
Sixth	10	3	0	0.002	78.563
Seventh	10	4	0	0.003	126.29
eighth	10	4	0	0.001	88.746
neinth	10	7	0	0.001	143.437

TEST

Table-2. Test initial state.



Trajectory of each MECABOT module:

Module 1: X: [4, 5, 6, 7, 8, 8]
Y: [4, 3, 2, 2, 1, 0]
Module 2:
X: [2, 3, 4, 5, 6, 7, 8]
Y: [6, 5, 4, 3, 2, 2, 1]
Module 3:
X: [3, 4, 4, 5, 6, 7, 8]
Y: [6, 5, 4, 3, 3, 3, 2]
Module 4:
X: [2, 3, 4, 5, 6, 7, 8]
Y: [5, 5, 4, 4, 4, 3, 3]
Module 5:
X: [6, 7, 8, 8]
Y: [5, 6, 5, 4]
Module 6:-
X: [6, 7, 8]

Y: [7, 6, 5]
Module 7:
X: [5, 6, 7, 8]
Y: [7, 7, 7, 6]
Module 8:
X: [5, 6, 7, 8]
Y: [9, 9, 8, 7]
Module 9:
X: [2, 3, 4, 5, 6, 7, 8]
Y: [9, 9, 9, 9, 9, 9, 8]

4. CONCLUSIONS

a) The algorithm designed initializes each of the variables in a random way, this allows to emulate the exploration activity for which the webots are designed, since it adapts to any distribution and quantity of obstacles.

b) The automatic configuration of each MECABOT to conform the chain is made by the shortest path because horizontal, vertical and diagonal movements are allowed, without touching any obstacle, which in this case, are the same MECABOTS. The routine that runs 2 squares to reach the goal allows an excellent assembly between modules.

c) The simulation was highly optimized, the execution time of the algorithm is quite short, which allows the user to see the MECABOT making decisions and executing trajectories in real time, this was achieved by tuning the FPS taking into account that affect the visualization or the correct functioning of the algorithm, it was also improved, by tuning the variable CFM that gives greater robustness against algorithm errors, without neglecting the visual quality of the environment. When executing the algorithm, it can be observed how fast the definition of intervals of the world trajectory is calculated, that the chain of MECABOTS must cross, so much so that the user does not perceive all the logical process that the algorithm performs in order to execute the homework. The algorithm is able to define the trajectory towards the goal, taking into account the obstacles, without affecting the best route for the MECABOTS chain.

d) According to the tests and results shown, it can be seen that an algorithm was developed that allows the automatic adaptability of the MECABOT robotic modules, taking into account the medium in which the compound agent formed is moved, since each of the objects in The world was randomly located in space and in all cases the result was satisfactory.

e) The performance of the algorithm when evaluating the chosen configuration with the input parameters allows the algorithm to analyze and order in real time, ideal state for this type of systems.

ACKNOWLEDGEMENTS

This work was supported by Davinci Investigation Group of the Military University Nueva Granada and it is associated to the investigation project ING-2634.

REFERENCES

[1] J. Seo, S. Gray, V. Kumar, y M. Yim. 2010. Reconfiguring Chain-Type Modular Robots based on the Carpenter's Rule Theorem. en Algorithmic Foundations of Robotics IX, Springer, Berlin, Heidelberg. pp. 105-120.

[2] Y. Fei y C. Wang. 2014. Self-Repairing Algorithm of Lattice-Type Self-Reconfigurable Modular Robots. J. Intell. Robot. Syst.75(2): 193-203, ago.

[3] S. Murata y H. Kurokawa. 2007. Self-reconfigurable robots. IEEE Robot. Autom. Mag. 14(1): 71-78.

[4] M. Yim, Y. Zhang, y D. Duff. 2002. Modular robots. IEEE Spectr. 39(2): 30-34.

[5] E. Ackerman. 2018. MIT's Modular Robotic Chain Is Whatever You Want It to Be. IEEE Spectrum: Technology, Engineering, and Science News, 12-jul-2016. [En línea]. Disponible en: https://spectrum.ieee.org/automaton/robotics/robotics-hardware/modular-robotic-chain-is-whatever-you-want-it-to-be. [Accedido: 16-feb-2018].

[6] HDSISymp2001Duff.pdf.

[7] N. Mathews, A. L. Christensen, R. O'Grady, F. Mondada, y M. Dorigo. 2017. Mergeable nervous systems for robots. Nat. Commun. 8(1): 439.

[8] Dtto is a 3D Printed, Self-Configurable Modular Robot, All3DP, 09-nov-2016. [En línea]. Disponible en: https://all3dp.com/dtto-3d-printed-modular-robot/. [Accedido: 20-feb-2018].

[9] Characterization of lattice modular robots by discrete displacement groups - IEEE Conference Publication. [En línea]. Disponible en: http://ieeexplore.ieee.org/abstract/document/5649934/. [Accessed: 16-feb-2018].

[10] Romanishin_3D_Mblocks.pdf.

[11] C. H. Belke y J. Paik. 2017. Mori: A Modular Origami Robot. IEEEASME Trans. Mechatron. 22(5): 2153-2164.

[12] The ATRON Self-reconfigurable Robot challenges and future directions Kasper Støy AdapTronics Group The Maersk Institute for Production Technology University. - ppt download. [En línea]. Disponible en: http://slideplayer.com/slide/3835627/. [Accessed: 21-feb-2018].

[13] J. Davey, N. Kwok, y M. Yim. 2012. Emulating self-reconfigurable robots - design of the SMORES system, en 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4464-4469.

[14] MOSS | Robotic Toys | Modular Robotics.

[15] J. Baca, S. G. M. Hossain, P. Dasgupta, C. A. Nelson, y A. Dutta. 2014. ModRED: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration, Robot. Auton. Syst. 62(7): 1002-1015.

[16] NASA - Snakebot. [En línea]. Disponible en: https://www.nasa.gov/centers/ames/news/releases/2000/00_66AR.html. [Accessed: 21-feb-2018].

[17] J. Howard. 2013. Self-Assembling Drones Take Flight in "Distributed Flight Array" VIDEO. Huffington Post, 24-jul-2013.

[18] Distributed Flight Array. [En línea]. Disponible en: http://www.idsc.ethz.ch/research-dandrea/research-projects/distributed-flight-array.html. [Accessed: 16-feb-2018].

[19] Pinpointing sources of water pollution with a robotic eel. [En línea]. Disponible en: https://actu.epfl.ch/news/pinpointing-sources-of-water-pollution-with-a-robo/. [Acessed: 21-feb-2018].

[20] M. J. Doyle, X. Xu, Y. Gu, F. Perez-Diaz, C. Parrott, y R. Groß. 2016. Modular Hydraulic Propulsion: A robot that moves by routing fluid through itself. en 2016 IEEE International Conference on Robotics and Automation (ICRA). pp. 5189-5196.

[21] Modular robotic eel hunts for sources of water pollution, Engadget. [En línea]. Disponible en: https://www.engadget.com/2017/07/25/epfl-robotic-eel-water-pollution/. [Accessed: 21-feb-2018].

[22] Self-Soldering Connectors for Modular Robots - Semantic Scholar. [En línea]. Disponible en: /paper/Self-Soldering-Connectors-for-Modular-Robots-Neubert-Rost/2e92ddcf2e7a9d6c27875ec442637e13753f21a2. [Accessed: 20-feb-2018].

[23] T. Tosun, J. Davey, C. Liu, y M. Yim. 2016. Design and characterization of the EP-Face connector. en 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 45-51.

[24] M. Cotera Berdugo. 2017. Simulación e implementación de una configuración de robot hexápodo utilizando sistemas de robótica modular para evaluar su locomoción. 2014 Cyberbotics Ltd 2014 Webots User Guide Release 743 Wwwcyberboticscom Cyberbotics.

[25] Webots documentation: Supervisor. [En línea]. Disponible en: https://www.cyberbotics.com/doc/reference/supervisor. [Accessed: 01-mar-2018].

[26] PATH FINDING - Dijkstra's and A* Algorithm's - Semantic Scholar. [En línea]. Disponible en: /paper/PATH-FINDING---Dijkstra%E2%80%99s-and-A*-Algorithm%E2%80%99s-Reddy/379d087b54850fa6d98c07c2e3bb66f51a109179. [Accessed: 01-mar-2018].