International Association of Scientific Innovation and Research (IASIR)
(An Association Unifying the Sciences, Engineering, and Applied Research)

ISSN (Print): 2279-0047
ISSN (Online): 2279-0055

International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Determine routes using an algorithm search with information in a simulated environment using Player / Stage and robot Pioneer

Michel Garcia[1], Cinhtia González[2], Sergio González[3]
Autonomous University of Yucatan Multidisciplinary Unit Tizimin.
Calle 48 Num. 207 x 31. CP 97700 Tizimín, Mexico

Abstract: *In this paper it implements a depth-first search algorithm performed in the C ++ programming language implemented on the Player / Stage, that allows the use of a robot Pioneer in a simulated environment. The program developed solves a maze that was represented as a binary tree; the results show a solution to the maze with path previously found, the robot travels the map from an initial state, and using its sensors to detect obstacles, the robot take a decision based on the information obtained with his sensors to solve map, same that allows you to reach a destination point.*

Keywords: *Player/Stage; Robot Pionner; Simulated Environment; Depth Search; Maze Search*

I. Introduction

Thanks to technological advances in recent years, today it is possible to build machines with several sensorial devices, with communication facilities, (in short, medium or long distance), and capable of perform different actions in different cases without need to an individual who controls or takes a decision, these machines use information to decide what action should be performed in each case, allowing to have autonomous robots. The decisions able to take these robots depend mainly on the data obtained from different sensors. Unfortunately, the development or acquisition of an autonomous robot can be expensive, but thanks to the current progress is possible deployments simulated environments like the Player / Stage [1], which is a free platform that provides a set of libraries that allows the programming and simulation of the behaviour of a robot in different types of environments, enabling to use all its sensors and / or actuators [2]. Following these advantages tools for simulation and robot programming, this project emerges in which an autonomous robot Pioneer 2-DX type is simulated [1], with the ability to solve the problem of finding a route between points in a maze, considering any of them as a starting point and another endpoint. To achieve this, it requires the use of sensors, actuators and a search algorithm. In this project will use the depth-first search algorithm. There are studies that use this tool using sensors and actuators to explore real-life terrain with multi-robots, these contain algorithms to perform successfully in the exploration of maps [2]. In [3] the problem of persecution-evasion consisting the realization of track addresses, that is where a robot chasing another; this problem is solved by using a modeling Markov decision processes. In [4] is stated that given a queue of delivery tasks prioritized objects, it is desired to find a sequence of actions for a mobile robot to perform these tasks efficiently, in that work a learning algorithm called Smoothed Sarsa is proposed.

II. Problem

The navigation of mobile robots is one of the major problems of artificial intelligence, it sets how a robot moves in a given environment, such navigation implies that the robot has the ability to interpret the sensor data and plan an path to a goal, the problem is defined as navigation. Given a starting point "A" reach the destination point "B" (B1, B2 ...) using the knowledge and the sensory information received from the robot [5]. The problem of navigation involves solving several problems, among these is the perception of the robot, that is must be able to interpret the sensor data to obtain useful information that will help determine its position in the environment in which it is located. Another problem is that planning must contain the robot, this means having the ability to decide how to respond to obstacles or crossroads and thus achieve the objective; likewise must have good control of their movement actuators to achieve the desired trajectory [5]. Such is the importance of navigation algorithms have been developed to increase the accuracy of the data collected by the sensors, so that the robot may know his current position with high precision. Because the amount of data to handle these systems, the programming must be as simple and efficient, the programming refers to the implementation that must have the robot in a programming language, defining the actions to be performed [6]. In [7], navigation maps are used to implement the simulation of several autonomous robots in a search and rescue operations, these robots store information from his sensors in the simulator and it keeps track of the actions of the robots for assess their performance, this implementation navigation and search missions dangerous rescue could provide benefits to humanity. For the problem of navigation in [8], route planning map dividing between robots is proposed, considering their travel times to minimize the time of complete navigation environment; all this is done by an algorithm that is designed to handle route planning and partition map simultaneously.

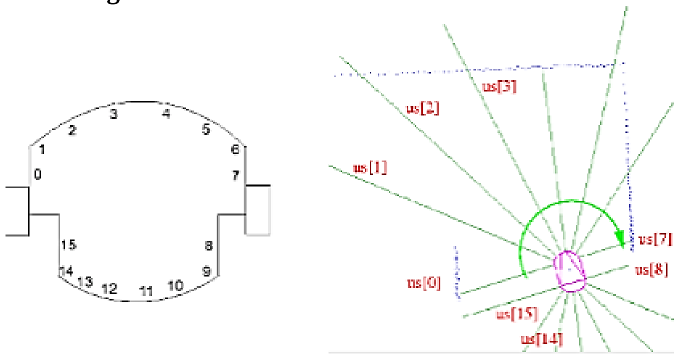
III. Requirements and tools for using the Pioneer robot

Player / Stage, is a free and freely distributable tool began as a project at the USC Robotics Research Lab in 1999 because of the need for an interface and a simulator for Multi-Robot Systems (MRS) [1]. This tool runs on UNIX-based platforms, for this project was implemented on the Linux distribution Ubuntu 11.10.

Player / Stage is a service layer and a simulator, offering facilities, transparency and speed for programming, control and simulation of robots like the Pioneer 2-DX, allowing the complexity involved in a transparent process to the developer. It is a service layer which allows to control the different components that contains a robot through its development interfaces, making it accessible the programming of a robot. It also contains drivers that simplify programming of the robot through its interfaces, allowing the implementation and configuration of various devices and sensors of the robot. The control programs communicates with Player through TCP sockets, reading data from the sensors, writing commands in the actuators, and configuring devices at the time.

For this project the ultrasonic sensor was used for its higher reliability to detect obstacles, also available in the simulation. Among the advantages of the ultrasonic sensor is the scope and quality of detection, which facilitates locating other available paths. The robot 16 has ultrasonic sensors distributed around the robot, in Figure 1 the distribution of sensor is appreciated.

Figure 1 Ultrasonic sensor in the Pioneer robot

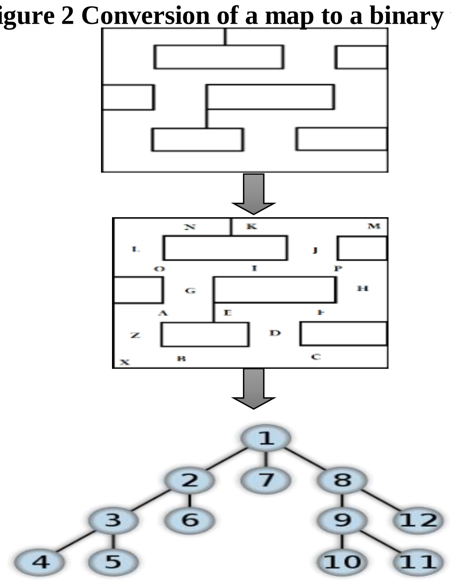


An important point in the programming of robots is being able to simulate the behavior, and this is where Stage does the work, through a graphical interface that simulates the real world through the use of an image that works like a map of the robot environment [6]; Stage has the ability to simulate various types of hardware and can simulate from one to hundreds of robots simultaneously on the same PC, including its sensors and devices. For the current project is simulated to a Pioneer 2-DX type robot. Among the advantages of the Stage, is the execution of algorithms reliability very close to reality [1].

IV. Implementation

To find the route from the start point to the end point, without the need to explore the surroundings, it was necessary to transform the environment map to a binary tree; to perform such activity, was defined a starting point in order to find the corresponding paths turning them into leaves of the tree, eventually all roads were referenced with values in order to generate the complete binary tree of the environment, it is noteworthy that the desired goal by the robot is specified in the generated tree. As seen in Figure 2 Naming and values to the map is observed.

Figure 2 Conversion of a map to a binary tree.



To find routes in a binary tree was necessary an algorithm that search each of its leaves to reach a solution, in this case the depth-first search algorithm was used. In the implementation was necessary to integrate the binary tree map coded in the C ++ language, values and names were assigned to the paths, facilitating the insertion of data into the program. In the code 1, shows the insertion of the nodes in the program. The functionality of this code is to accommodate the nodes by assigning a name and a value that allows creating a binary tree.

Code 1: node insertion

```
case 0: insertar( arbol, 1000,'X');
case 1: insertar( arbol, 950,'Z');
case 2: insertar( arbol, 900,'A');
```

Once entered the names and values of all nodes required by the map, it was required to structure nodes in order to complete the tree. The function shown in Code 2 was implemented which allows lower nodes go to the left and higher nodes go to the right, resulting in the complete tree.

Code 2: Generation of the tree

```
void insertar(ABB &arbol, int x,char i){
        if(arbol==NULL){
        arbol = crearNodo(x,i);
        }
        else if(x < arbol->nro)
        insertar(arbol->izq, x,i);
                else if(x > arbol->nro)
                insertar(arbol->der, x,i);
}
```

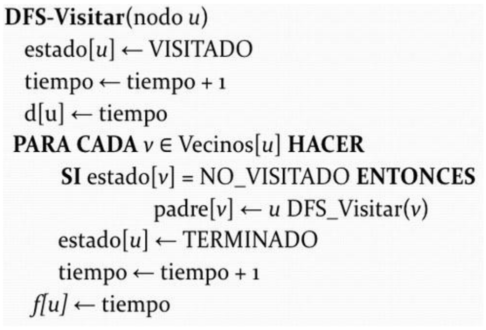
Once structured the nodes on the tree, was necessary to implement the algorithm to find the path to the target or goal, this algorithm is known as depth-first search, which part of your coding is shown in code 3.

Code 3: Depth Search Algorithm

```
void enOrden(ABB arbol){
        if(arbol!=NULL){
        if(arbol->Inf=='M'&&bandera!=1){
        bandera=1;
        camino2[tam]=arbol->Inf;}
        if(bandera!=1){
        camino2[tam]=arbol->Inf;
        tam++;
        }
enOrden(arbol->izq);
enOrden(arbol->der);
tam--;}
}
```

The algorithm allows to visit all the nodes visiting each of recursively that is, not all children nodes will be visited consecutively, they will travel in depth until to reach a end node or leaf node, if there are no more nodes to visit, will do a backtracking, the process is repeated with each of the brothers and the processing nodes. In the figure 3 the operation is shown. Upon completion, the algorithm throws us the path to the goal.

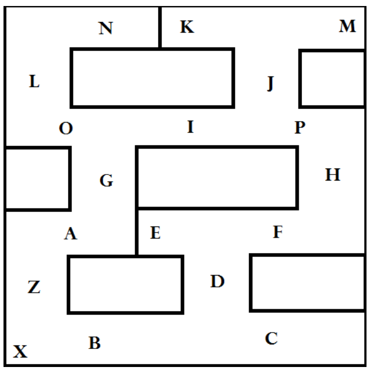
Figure 3 Depth Search pseudocode



```
DFS-Visitar(nodo u)
  estado[u] ← VISITADO
  tiempo ← tiempo + 1
  d[u] ← tiempo
  PARA CADA v ∈ Vecinos[u] HACER
      SI estado[v] = NO_VISITADO ENTONCES
              padre[v] ← u DFS_Visitar(v)
      estado[u] ← TERMINADO
      tiempo ← tiempo + 1
    f[u] ← tiempo
```

Subsequent to implement the algorithm. The programming of the sensory part of the robot was necessary, the sensor has 16 positions to use for which it was decided to use 8 of them, of which a vision of the left side was obtained, forward and right side beginning from position 0 to position 7, Figure 1 shows the configuration of the position of each of the sensors.
The operation of the entire program consists of a combination of programming the depth search algorithm and programming the ultrasonic sensor.
Once the algorithm finds the path, the robot starts in an initial state, subsequently continues to advance until the sensor detects several paths or obstacles in the map, then the program asks the next option that showed the depth search algorithm, this process is carried out until the robot reaches its final state.
Tests were performed on simulated environment using a maze map as shown in Figure 2, this map assigned names for each path, as shown in Figure 4.

Figure 4 Naming paths



Once designed the map with names, values are placed to make a binary tree, considering that the initial state is the "X" and the final state is "M".
According to the algorithm explained above, running the algorithm will find the following path: X, Z, A, G, I, J, M, this is the route that runs through the robot in the simulator.

V. Test and Results

To run the program first the .cc file was compiled with the sentence: *g ++ -o outfile `pkg-config --cflags playerc ArchivoFuente.cc ++` `pkg-config` --libs playerc ++*, followed by the load of the map image with the sentence: *player CodigoMapa.cfg*.
The application was executed approximately 40 times of which about 32 of those occasions reached the goal. When the system did not succeed was due to the characteristics of the simulator, because when you run some other concurrent application Player was not provided very accurate readings of the sensors and the execution of the angle of rotation necessary for the proper execution of the route found was incorrect.
When the robot reaches the goal it did so almost reflective, ie the trajectories described by the robot were almost identical.
During testing the application was submitted to various experiments, among the most important change is the goal. The program succeeded in all cases meet the goal, with the obvious difference of the time it took for to obtain the route from the starting point to the goal.

VI. Conclusions

The use of depth-first search algorithm is useful in event that has all the map information. This means that the map can be loaded as a tree that can be solved with depth-first search technique. The application developed solves the map to find the route to follow, it then runs the actions towards the goal, so that the robot does not need to have stored in its memory all the map information. This ensures efficiency in the time when the robot reaches its goal, because the resolution of the map is not solved by the robot and this one just follows a number of specific instructions.
One of the problems facing this implementation is the conversion of a binary tree map. This also creates a problem since adaptation of the a binary tree map is necessary to identify the target with the letter "M".
The results showed that the robot reaches the goal in 95% of the time. But the application generated a 100% effectiveness when determining the route to the goal, the occasions when the robot do not reaches the goal is due to problems of the simulator.
One of the future work will be to implement another algorithm using heuristics for finding routes, same that allows finding optimal routes, another future work could parallelized the process, and that with greater processing capacity would be developed an implementation for non-binary trees.

VII. References

[1] P. Gerkey Brian, T. Vaughan Richard, Andrew Howard, "The Player/Stage project: Tools for Multi-Robot and Distributed Sensor Systems".

[2] K.S. Senthilkumar, K. K. Bharadwaj, "Player/Stage Based Simulator for simultaneous multi-robot exploration and terrain coverage problem", Vol. 2, Octubre 2011.

[3] Michel Garcia G., Cinthia González S., Eduardo Morales M., Enrique Sucar S., "Modelado y solución del problema de persecución - evasión en robots móviles utilizando Iteración de valor", Octubre 2007.

[4] Deepak Ramachandran, Rakesh Gupta, "Smoothed Sarsa: Reinforcement Learning for Robot Delivery Tasks", Japón, 2009.

[5] Alberto Ortiz, "Navegación para Robots Móviles", Departamento de Matemáticas e Informática.

[6] Tomas Rebollo Balaguer, "Diseño e implementación de una capa de software para el control de robots mediante Player/Stage", 2010.

[7] Robert L. Dollarhide, Arvin Agah, "Modeling and simulation of autonomous robot search teams", 2002, Vol. 42, pp. 1739-1761.

[8] Muzaffer Kapanoglu, Mete Alikalfa, et al., "A pattern-based genetic algorithm for multi-robot coverage path planning minimizing completion time", May 2010.

Rich Y Knight K., "Inteligencia Artificial", 1994, McGraw Hill, Segunda Edición.

VIII. Acknowledgments

We thank the Autonomous University of Yucatán and the Faculty of Mathematics from all facilities in the accomplishment of this work.