AUTONOMOUS NAVIGATION ROBOTIC SYSTEM TO RECOGNIZE IRREGULAR PATTERNS

Keywords: Autonomous Mobile Robots, Learning Algorithms, Artificial intelligence.

Abstract: This paper presents an approximation of navigation problem under unknown environment using reinforcement learning. The motivation is to represent a robot that can move in a world with streets and intersections. Each intersection has a different quantity of streets (irregular blocks). The selected algorithms were Q-Learning and Value Iteration. The robot was programmed only with Q-Learning and we developed a simulation with both of them. This simulation allows making comparisons in order to determinate in which situation each algorithm is appropriate.

1. SYNOPSIS

This paper presents a particular application for unknown environment using two techniques of reinforcement learning. These algorithms allow an autonomous agent or robot, to reach a goal by learning the environment. In our case, the world is formed by blocks of irregular geometric forms. The robot does not know a priori the world but he learns to identify it. The robot only knows previously the location of the goal, given in terms of coordinates (x, y).

The reinforcement learning algorithm selected for the robot was Q-Learning. This algorithm consists on rewarding or penalizing each possible action that the robot executes. Given a policy, the robot is able to reach a goal, learning by trying and testing.

The simulation was developed using Q-Learning and Value Iteration in order to test these algorithms in different worlds. It allows comparing them under different situations, either changing the form of the blocks of the world, or changing the position of the goal.

The robot was built with a Lego Mindstorm 2.0 kit with two light sensors and one rotation sensor. The selected language for robot application was Java under Lejos platform and the simulation was developed in Visual Basic.

1.1 Justification

The problem arises when an agent need to be located in an unknown environment. For example, in Caracas, Venezuela, most of blocks are rectangular and the movements are given in right angles. However, in other places, blocks can be of irregular forms and it is possible to end up losing the sense of the orientation. If an agent do not know to move in places where blocks are irregulars, it is necessary a new learning process that can make the task of recognition of these patterns. It can be able to go from a point to another without to get lost.

2. REINFORCEMENT LEARNING

The reinforcement learning uses recompenses and penalizations when the agent executes an action in order to reach the goal. In reinforcement learning an agent interacts with their environment. This interaction allows the agent, based on sensor inputs, chooses an action to be carried out in the world.

The learned behavior contains the robot's implicit world model. By using reinforcement learning it is not needed to have examples to build and to validate the behavior (like supervised learning). The behavior is synthesized using a scalar (reinforcement) as the only source of information. This information allows evaluate the action. The agent receives positive, negative or null reinforcements according situations and action selected. There is not separation between the learning phase and the using phase.

In this work, the world is inaccessible a priori to the robot (it must learn the world), recompenses are received in any state or situation, and the robot is a passive apprentice (it does not alter the world).

2.1 The reinforcement and value functions

In reinforcement learning must be defined a reinforcement function. This is a function of the future reinforcements that the agent searches to maximize. In other words, a relationship exists between the couple state/action with the reinforcements. After carrying out an action in a state, the agent will receive some reinforcement

(reward) given as a scalar value. This reinforcement defines the value function. The agent learns how to maximize the sum of the received reinforcements when begins from an initial state and proceeds to a terminal state.

The value function is based on a field of the mathematics called dynamic programming. To explain it, some notations is needed: $V^*(x_t)$ is the optimal value function, where x_t is the vector of states; $V(x_t)$ is an approximation of the function value; γ is the discount factor in the range of [0,1] that causes that the immediate reinforcement has more importance than the future reinforcement. In general, $V(x_t)$ can be initialized with random values and it does not contain information about the optimal value function $V^*(x_t)$. This means that the approximation of the optimal value function in a given state is similar to the value of that state given by $V^*(x_t)$ plus some error in this approximation. The following equation (1) express this idea:

$$V(x_t) = e(x_t) + V^*(x_t) \qquad (1)$$

where $e(x_t)$ is the error in the approximation of the value, in this state, in the time t.

The value of the state x_t follow a policy that is the sum of the reinforcements when it begins of the state x_t and carrying out good actions until a terminal state is reached. For this definition, a simple relationship exists between the values of the successive states x_t and x_{t+1}. This relationship is expressed by the Bellman equation (2). The discount factor γ is used exponentially for decrement the weight of the reinforcements received in the future.

$$V(x_t) = r(x_t) + \gamma V(x_{t+1}) \qquad (2)$$

The process of the learning is the process to find a solution for the equation (2) for all the states x_t, through the space of states. If there are not changes in the state values, then the state values have converged.

2.2 Value Iteration

One can find an approximation to optimal value function evaluating for all possible state since x_t. In equation (3), u is the action performed in state x_t and cause a transition to state x_{t+1} and $r(x_t,u)$ is the reinforcement received when performing action u in state x_t.

$$V(x_t) = max_u\,(r(x_t,u) + \gamma V(x_{t+1})) \qquad (3)$$

The figure 1 illustrates this process.

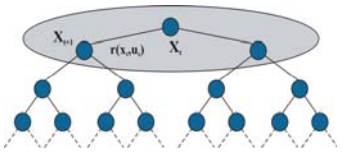

Figure 1: Tree of states for Value Iteration

The previous figure describes how update the value in x_t. Specifically, there are two possible actions in the state x_t, and each one of these actions directs to different states successors x_{t+1}. To update in Value Iteration, it should find the action that returns the maximum value (BFS algorithm). The only way to achieve this is by executing an action and calculating the sum of the received reinforcement and the approximate value of the state successor x_t.

2.3 Q-Learning

Q-Learning is another extension to the traditional dynamic programming that solves the problem more simply. Value Iteration requires finding the action that the maximum prospective value returns (the sum of the reinforcement on all the possible states successors for the given action).

Q-Learning differs from Value Iteration in that it does not require in a given state each action be performed and the expected values of the successor state be calculated. While Value Iteration executes BFS, Q-Learning takes a single step sample. It only needs to be the maximum value in the new state to have all the necessary information to revise the prediction associated with the executed action. Q-Learning does not require to calculate on all the possible successor states. This process is demonstrated in figure 2:

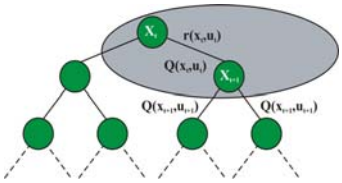

Figure 2: Tree of States vs. Actions

In Q-Learning each situation-action has a value (called Q-value). In each state, there is a Q-value associated with each action (a matrix). The definition of the Q-value is the sum of the received reinforcements when the action is executed following a policy. So Q-Learning represents values with a matrix instead of Value Iteration that represents it with a vector (only a value for each state).

The Q-value of a state should approximate the maximum Q-value in the given state. In this case, it is easy to derive the equivalent of the Bellman equation (4) for Q-Learning:

$$Q(x_t,u_t) = r(x_t,u_t) + \gamma \max_{ut+1}(Q(x_{t+1},u_{t+1})) \quad (4)$$

After the execution of the action for the robot in the real world, a reinforcement function provides a reinforcement value. It can take +1,-1 or 0 values. It is used by the update function to adjust the recompense value (Q) associated to the situation-action in the robot's memory.

2.3.1 Q-Learning Algorithm

In this application, the algorithm was modified including ß factor. β represents the learning rate and it should be bigger than 0. The algorithm steps are:

1. Initialization of the robot's memory: for all the situation-action couples, the Q-values are zero. It also could be initialized with random values.
2. repeat
   a. *x* is the situation of the world.
   b. The evaluation function selects the action to be executed: $x = Max(Q(x,u'))$
   where *u*' represents any possible action. The selection process depends on the distance to the goal.
   c. The robot executes the action *u* in the world. Be *r* the recompense (*r* can be 0) associated with the execution of that action in the world.
   d. Updates the robot's memory:

$$Q_{t+1}(x,u) = Q_t(x,u) + \beta\,(r + \gamma \max(Q_t(x', u')) - Q_t(x,u))$$

where x' is the new situation after having been executed the action in the situation *x*, u' represents any possible action.

This algorithm was implemented in Java: LejOS (Lego Java Operating System) and is executed by the robot when it navigates in the world.

3. APPLICATION DEVELOPPED

3.1. Analysis

This work was developed in order to ask the following questions:

(1) Which characteristics should have the world where the robot is unwrapped?
The idea is to learn how can be guided a robot in a world whose blocks form are ignored by the autonomous entity.

(2) What reinforcement learning algorithms should be implemented?
Q-Learning and Value Iteration are the most popular algorithms. It should be interesting compare these algorithms under different constraints.

(3) How do apply Q-Learning and Value Iteration to our application?
In order to navigate problem, the robot is rewarded if this comes closer to the goal and it is punished if it moves away from him.

(4) Which operating system (firmware) is appropriate to program the robot?
There are four possible operative systems to program the robot: BrickOS (well-known previously as legOS), NQC, the Robotic Invention System (RI), and LejOS. The last one is based in JAVA, so highly Object Oriented, the maintenance is easier than in a language structured as C and it is also available for a wide variety of platforms.

3.2 Design

It should be convenient to design and to prove different types of physic structures for the autonomous entity (robot), in order to define and to build the environment and finally to adjust the algorithms Q-Learning and Value Iteration.

3.2.1 Robot Design

The first problem that was presented was the accuracy of the movements and the turns, since the robot should make turns on his owns axis and movements along the streets or roads. If errors are present in the turn, the robot takes a wrong road, harming in a significant way the execution of the algorithm.

Due to the inaccuracy in the movement, we used a dual differential drive, such as it shown in the figure 3.

Figure 3: Transmission system with dual differential drive

In this system, a motor takes charge of moving the robot straight line (forward or back), while the second motor takes charge of carrying out the turns (toward the left or toward the right), and one of the motors should only work at the same time, both cannot work at the same time. The advantage of using this system resides in that both wheels work to the same speed maintaining a uniform movement.

Once resolved the problem of the accuracy of the movements, initially it was planned to use caterpillars, but these produced an undesirable effect in the turns, since these spread to move toward the sides. After proving with several types of configurations we made the decision of using two wheels behind and a stabilizer piece in front of the robot. This piece acts as a ski, minimizing the close contact with the floor and achieving a stable movement as much in the turns as in the right movements.

In order to capture the events of the environment, a study was made to define what kind of sensors would be used for such an end. Initially it was planned to use three ultrasonic sensors, placed one in the robot's front part, and the other ones two in the lateral sides of the same one. However, serious inconveniences arose since with the sensors the readings, in many cases, they didn't reflect the real situation, for example, when it happens a phenomenon called speculate reflection.

It was adopted two sensor of light under the robot pointing toward the floor and a rotation sensor. This solves part of the problems mentioned previously, as the recognition of roads and intersections. The sensor of light maintains adjusted the robot to the road because that, if the road is of a certain color, for example blue, and then it detects another color, for example white, is adjusted to follow the road. The rotation sensor allows to know how many degrees the robot has rotated, being an important element for the robot's navigation. It could obtain an error of ±10º.

Finally the last sensor of light was placed pointing to the floor in the robot's front part, which allows to detect in a quick and simple way how many roads are in each intersection. Lastly, the robot's physical construction is:

Figure 4: The robot's final version

3.2.2 Environment

Once concluded the robot's physical structure, we proceeded to the construction of the world. With the elimination of the sonar, we decided to eliminate the walls and to identify background, roads and intersections with the white, blue and black color, respectively.

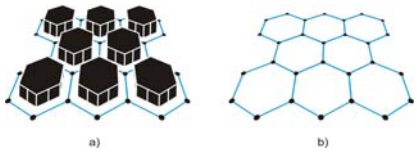

Figure 5: World types. a) With walls. b) Without walls

3.2.3 Q-Learning adaptation

Firstly, it is necessary to learn how to arrive to a specific point of the map from a beginning point. In each intersection it should be decided among a series of roads, some move away the robot of the goal, other brings near it and some even maintain the same distance from the goal. If an action in that state brings near the robot of the goal, this action it should be rewarded, while if the robot goes away, the action should be penalized. The autonomous entity should learn how to arrive to its destination choosing the actions that offer a bigger recompense.

Q-Learning requires to elaborate a matrix (mainly called Q-Matrix) of actions against states. The actions in this case are: rotate a specific angle and to move straight . The figure 6 shows some of the possible actions that the robot can take in an intersection with three ways:

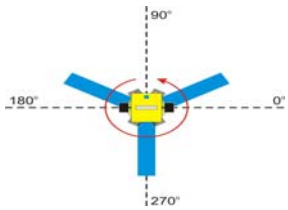

Figure 6: Actions that the robot can carry out in an intersection type

In this point, the robot can rotate toward the road that is at 30°, 150° or 270° with regard to the horizontal. In the Q-Matrix, these actions are denominated by the value of the angle that should be rotated.

The states represent situations that can happen in the robot's interaction with the world and they can be directly the values read by the sensors, combinations or derived abstractions of the readings of the sensors or robot's internal representations.

The proposed design of states was one where the robot built a coordinated system where the origin is the starting point (Figure 7) and the coordinates of the goal are already fixed and known previously by the robot. Using these references, and trigonometric calculations, the distance is calculated between the robot and the goal or objective.

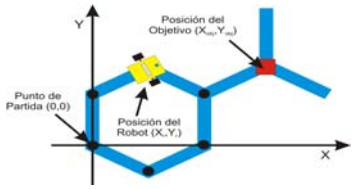

Figure 7: System of coordinates used by the robot

It is also necessary to know where is the robot with respect to the goal (if it is above, below, to the right or to the left of the goal). To obtain that information, the coordinates of the goal are subtracted with the robot's coordinates, for example, if the subtraction of the abscissas and the coordinates are negative, the robot is above and to the left of the goal (quadrant I of the figure 8).

The combinations between the quadrants and the differences among the distances from the robot to the goal define the states. In the table 1, all the possible states are shown where d_{ant} indicates the distance from the robot to the goal when it was in the previous intersection. d_{act} indicates the distance from the robot to the goal in the current intersection.

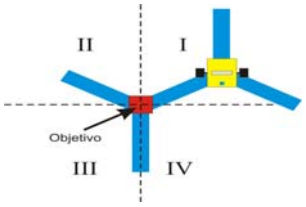

Figure 8: The robot's position with regard to the goal

State	Quadrant	Distance
0	1	$d_{ant} < d_{act}$
1	1	$d_{ant} > d_{act}$
2	2	$d_{ant} < d_{act}$
3	2	$d_{ant} > d_{act}$
4	3	$d_{ant} < d_{act}$
5	3	$d_{ant} > d_{act}$
6	4	$d_{ant} < d_{act}$
7	4	$d_{ant} > d_{act}$

Table 1. Possible States for the Q-Matrix

Finally, the Q-Matrix (table 2) was built using the states mentioned previously, and they took the actions as the angles from 0° up to 360° numbered of 10 in 10 (rotation sensor precision), that is to say, 0°, 10°, 20°, etc. The lines are the states and the columns are the actions.

State\Action	0	10	.....	350	360
0	0,48	1	.....	0,93	0,23
1	0,70	0,60	.....	0,67	0,37
2	0,97	0,45	.....	0,23	0,87
3	0,65	0,74	.....	0,62	0,34
4	0,10	0,76	.....	0,34	0,73
5	0,29	0,56	.....	0,89	0,66
6	0,69	0,77	.....	0,83	0,65

Table 2. Q-matrix

Each state has associated a series of angles that represent the possible actions. For example, if the robot arrived to an intersection that brought near it to the objective and it is in the quadrant 3, the state that represents this action is the state 5 (to see table 1 of states). Once identified the state, determines the action that it will take looking for the biggest value in the line for that state. In the table 2, it can be appreciated that the maximum value for the state 5 corresponds to the angle of 350° for what the action that the robot will take will be the one of rotating up to 350° to move later on straight line until the next intersection. These value used to determine the maximum, Q-values, represent the utility of a couple (state/action).

Once done this and coming to the following intersection, the robot identifies the state again. It calculates the recompense of the action taken in the previous state. A positive recompense takes place when the robot executes an action that brings near it to the goal, rewarding it. While a negative recompense takes place when instead of coming closer to the goal this goes away, for what the executed action is penalized (figure 9). With the calculated recompense, the Q-value is updated (value in the Q-Matrix) corresponding to the couple (previous state-action) modifying the values that can affect to future decisions for that state.

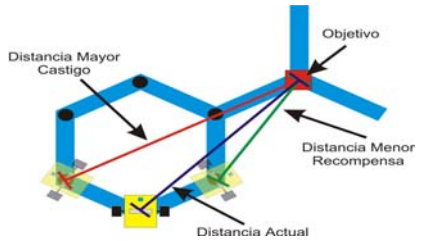

Figure 9: Rewards of the possible actions that the robot can take

3.3 Implementation

Q-Learning simulation uses the following classes:

- **RotationNav.java**: carries out the robot's movements and the turns, to move from a point to another, etc. Uses the rotation sensor to know what angle is with regard to the horizontal.
- **Qlearning.java**: calculates the robot's states, to reward the actions taken in each state and of storing in the Q-Matrix, all that has to do with the robot's learning.
- **Huron.java**: manages and coordinates all the calls to the other two classes explained previously.

Value Iteration and Q-Learning algorithms were programmed in the simulation, to be able to make the comparisons among them. It is pertinent to expose first how Value Iteration can be applied to the navigation problem. In Value Iteration the taken actions in a state always take the same successor state, while in Q-Learning, the taken action in a state could be anyone of the other states that is not always the same one. For this reason, we had to change the approach of actions and states so that Value Iteration could be applied.

The approach of actions is the same that was used for Q-Learning, but the states were implemented in a different way. It was opted to represent the states like the intersections of the main map, identified by whole numbers. Once defined the states and actions, it is necessary to proceed to elaborate the base of knowledge. The algorithm only needs to know the utility (V-Value) for state:

	1	**2**	**3**	**….**	**49**	**50**
1	V(1)	90	30	….	-1	-1
2	270	V(2)	-1	….	-1	-1
3	190	-1	V(3)	….	-1	0
….	….	….	….	….	….	….
49	-1	-1	-1	….	V(49)	0
50	-1	-1	-1	….	180	V(50)

Table 3. Knowledge base used

The approach of rewards stays similar to Q-Learning, if the robot comes closer to the goal this it is rewarded but if the robot goes away it's penalized.

The application of Value Iteration is relatively simple. When the autonomous entity arrives to a new intersection, it evaluates the different roads that can take in that an intersection. Then it evaluates how is the reward that receives for each possible road, added with the V value from the state, which arrives if it executes that action. Once carried out all these operations, the road is chosen whose sum is the biggest, V-Value of the current node is substituted with that of the selected road and takes this road, and repeat all the steps mentioned previously in this paragraph until it finds the goal.

Because the learning is carried out in each state, the robot considers that he has learned in that state when the value previous V doesn't differ of the value current V, the error among them is zero.

3.4 Tests

In this last phase was carried out successive tests for the detection of error. The robot's acting was observed in the environment and the behavior of the algorithm Q-learning applied for the navigation.

Nevertheless, the principal objective in this phase is to establish the comparisons among Value Iteration and Q-Learning, using the simulation. It is important to mention that a iteration begins with the robot in the starting point and it concludes when this arrives to the goal. Each run can have associated a finite number of iterations.

The Q-Learning reinforcement learning algorithm showed to be a good option for the navigation problems. In several runs with the built robot it could be observed that the algorithm was able to reach the goal most of the times. There were movements those performance was affected by random external events and the errors that were presented in the readings of the sensors of light.

With regard to the simulation, the algorithms Q-Learning and Value Iteration, one could observe the behavior of each one in an ideal world. In this way, we can carry out comparisons between both algorithms, avoiding the problems with the inaccurate of the sensors. Also, modifications can be made in certain parts of the algorithms, without altering the logic of them.

The quantity of iterations indicates how many times each one of the algorithms were executed.

Test 1

There was carried out a run of 20 iterations, maintaining the goal fixes, with each one of the two

algorithms to see how it was the learning curve in both cases. The obtained graph was the following one:

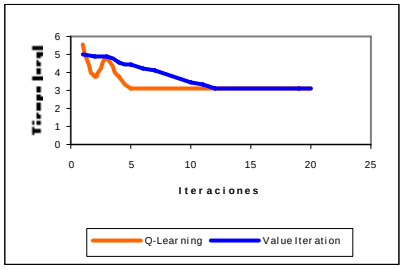

Figure 10: Learning curves of Q-Learning and Value Iteration

Here one can observe that Q-Learning converges in quicker way than Value Iteration, that is to say, it takes less learning time to arrive to the goal. Even though Value Iteration takes more time to learn, both of them find efficiently the goal with the movement policy.

Test 2

Since the Q-Matrix can be initialized with random values or with values in zero, they were carried out two runs where one of them was initialized with random values and the other one with zeros. The result was the following one:

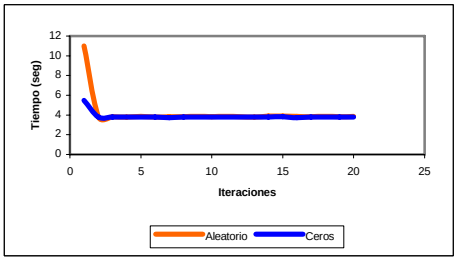

Figure 11: Learning curves of Q-Learning

You can infer of this graph that don't care values with which the Q-Matrix is initialized, since always it will converge in a similar way. The only thing that could change is the time that takes in learning how to arrive to the goal in the first iterations.

Test 3

Were carried out 2 runs of the Q-Learning algorithm of 40 iterations each one, changing the position of the goal after each 10 iterations and maintaining it fixed during those 10 iterations. For the first run we worked with the matrix initialized with random values and for the second run the matrix was initialized with zeros. The resulting graph was the following one:

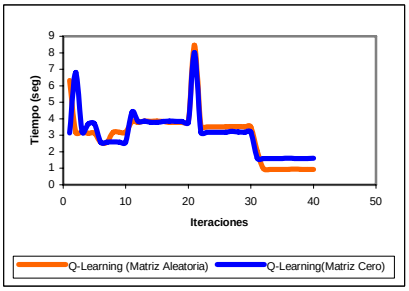

Figure 12: Learning curve of Q-Learning varying goal

Here we can observe that the behavior of Q-Learning for both cases is similar, they are able to almost establish the same time of arrival to the goal. The peaks represent the point where the goal was changed position and due to this, the algorithm takes more time for the learning when trying to adapt to the new change.

Test 4

Were carried out 2 runs of 40 iterations each one, changing the position of the goal each 10 iterations. For the first run we worked with the Q-Learning algorithm and for second one, the algorithm Value Iteration. The resulting graph was the following:

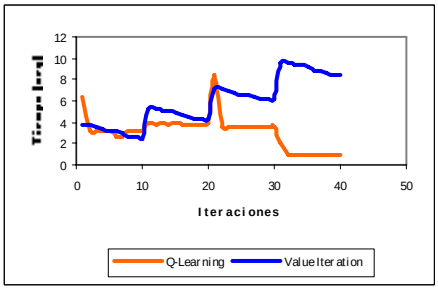

Figure 13: Learning curves of Q-Learning and Value Iteration varying goal

As you can observe, Q-Learning adapts to the changes in a quick way and like the previous graph. However, Value Iteration, once had concluded the learning of the route to the goal, when it change, the new time that takes is equivalent at the time that took in arriving to the position previous of the goal plus the time that needs learn how to arrive to the new position of the goal.

It is important to notice that Value Iteration, keep the route in a state's graph. It is affected critically if it is forced to go by a node or intersection already visited. Each intersection has assigned an action that was been of the previous learning, what causes to take the same road every time that arrives to that state. It does not carried out the learning process for

that intersection, retarding the arrival to the new position of the goal.

Test 5

Due to the previously exposed problem for the Value Iteration algorithm and for the time that takes learning how to arrive to the new goal. It was decided to make a modification that consisted on eliminating the condition of learning end, that is to say that is carried out the learning process in each node or visited intersection. For this case it were also carried out 2 runs changing the position goal, of 20 iterations each one, using for the first one the Q-Learning algorithm and for second one the Value Iteration algorithm. The resulting graph is shown next:

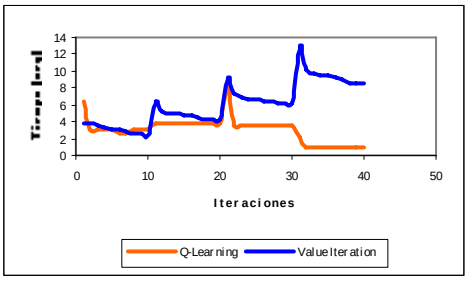

Figure 14: Learning curves of Q-Learning and Value Iteration modified varying goal

We observed that there was not a significant change in the behavior of Value Iteration, because it follows the same behavior patterns of the previous case, only that the learning was made in all moment, in each intersection visited or not visited.

Test 6

Lastly, there was carried out a second modification in the Value Iteration algorithm that consisted in reinitialize the graph every time that the position of the goal was changed, maintaining the condition of learning end. Were also carried out 2 runs of 20 iterations each one, changing the position of the goal and using for the first run the algorithm Q-Learning and for second one the algorithm Value Iteration.

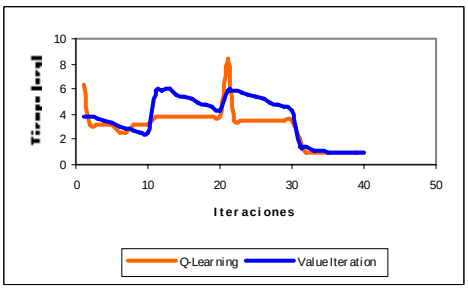

Figure 15: Learning curves of Q-Learning and Value Iteration modified varying goal

For this case, the algorithm Valued Iteration adapted better to the changes of position of the goal, but that in comparison with Q-Learning, it continues taking more time of learning than the one used by Q-Learning. The disadvantage of this is that no longer keeps its knowledge if the robot should travel the world again.

It is necessary to highlight that the Value Iteration algorithm always search the shortest route, while Q-Learning only tries to arrive to the goal according to a policy of movements product of the learning. Occasionally, Value Iteration visited less intersections than those visited by Q-Learning

4. CONCLUSIONS

Along this work, we studied different aspects of the artificial intelligence applied to the area of the learning, specifically the reinforcement learning. They allow solve a great variety of problems where learning is an important factor to search a solution.

The use of the kit Lego MindStorm was appropriate for the construction of the robot's prototype, because it allows build easy and quickly a great variety of adaptive robots' models to diverse problems. In particular, it is necessary to highlight that the rotation sensor possesses a remarkable accuracy in the measured values. In contrast with the light sensors whose readings were affected by external factors as the environmental light. A disadvantage of the kit is the quantity of inputs or sensors that has the robot. It allows connect only three sensors at the same time, limiting the functionality of the robots and the complexity of the program that is implemented. Also, it possesses a limited memory (32Kb), what do not allow implement very extensive programs. Also, the firmware, occupies space in the memory, diminishing even more the capacity to store user's programs.

With regard to the operating system (firmware), the election of LejOS was convenient for several reasons. In the first place, this only occupies 16 KB of the memory, leaving available the remaining 16KB of memory for user's programs. BrickOS (legOS) occupies 20 KB, leaving 12 KB of memory for user's programs. In second place, LejOS it is a firmware based on the programming language Java, inheriting the potent capacities that this language provides.

The navigation problem to find a goal is possible to solve using a reinforcement learning algorithm. Other types of learning techniques exist, as the evolutionary learning that involves the use of genetic

algorithms and that can be used to solve navigation problems. These algorithms are slow and they only allow approximate to sub-optimal solution, without any guarantee of the convergence to the best solution. They do not operate on real time execution, being a restrictive factor for the implementation of this type of problem.

Value Iteration, as Q-Learning, is based on the principle of the dynamic programming, which allows carry out an effective learning through recompenses and penalizations. The advantage of using Value Iteration is that the found solution tends to be, in most of the cases, better than the solution found with Q-Learning.

The main disadvantage of the Value Iteration algorithm is that it only can be applied in action-state schemes where an action taken in a given state, leads the same successor state. If the goal changes position, the algorithm takes much more time, since all the roads that have been taken and then learned by the robot, these will always be taken even still when the goal is located in some other point.

With regard to Q-Learning, the main advantages of using this algorithm are that if the goal changes position, Q-Learning adjusts its learning efficiently, being able to always arrive to the goal. Besides if the Q-Matrix is initialized with random values, the robot can experience other roads that enlarge his learning. This is because the algorithm looks always for the maximum Q-value for all possible actions in a state. Finally, the learning curve tends to converge in quicker way that Value Iteration, although in a less uniform way.

The main disadvantage of using Q-learning is that the solution found is not always the best one, although it tends to be very near of it.

In this point one could wonder under what conditions one of these two reinforcement learning algorithms should be chosen? The answer comes given depending on the situation. If the time to arrive to the goal is not a problem, you can apply Value Iteration, otherwise Q-Learning is the best option. However, for both cases, the route that finds Value Iteration is similar to the route that finds with Q-Learning, varying very little with respect to the other one.

When programs are designed based on reinforcement learning algorithms, it is necessary to define and to design in a detailed way the states, actions and the reward policy, since these factors play a very important role in the operation of the algorithm. If some of these factors flaw, the acting of the algorithms can be seriously affected, or worse, it could not arrive to any solution.

REFERENCES

Bagnall, B. 2002, CORE Lego MindstormsTM Programming. Edit. Prentice Hall PTR. New York, United States of America.

Ferrari, G., Gombos, A. Hilmer, S., Stuber, J., Porter, M., Waldinger, J. and Laverde, D. 2002, Programming Lego MindstormsTM with Java. Edit. Sysgress. United States America.

Rich, E. and Knight, K. 1994, Artificial Intelligence. Second edition. Edit. McGraw Hill. Madrid.

Russel, S. and Norvig, P. 1996, Artificial Intelligence A modern approach. First edition. Edit. Prentice Hall, 1996.

Pressman, Roger S. 1998, Engineering of the Software a practical focus. Fourth edition. Edit. McGraw Hill. Madrid, Spain.

Carrasquero Z., Oscar H. and McMaster F., Eduardo 2002, Design and a robot's construction with the module RCX 1.0 for not predetermined worlds (Thesis of Engineer in Computer, Catholic University Andrés Bello).

Bagnall, Bryan 2001, LejOS: Java for the RCX. [web page on-line] Consulted in 10/5/2003 Available in http://lejos.sourceforge.net /.

BrickOS Home Page. [web page on-line] Consulted in 2/5/2003 Available in http://brickos.sourceforge.net /.

Dartmouth College Computer Science Department (2001). Robbery-Rats Locomotion: Dual Differential Drive. [web page on-line] Consulted in 20/5/2003 Available in http://www.cs.dartmouth.edu/~robotlab/robotlab/courses/cs54-2001s/dualdiff.html

Gross M., Stephan V. and Boehme J. 1996, Sensory based robot navigation using self-organizing networks and Q-Learning. [document on-line] Consulted in 12/9/2003 Available in http://citeseer.nj.nec.com/gross96sensorybased.htm

Mance E. Harmon and Stephanie S. Harmon (s.f.). Reinforcement Learning: A Tutorial. [document on-line] Consulted in 20/9/2003 Available in http://www.nada.kth.se/kurser/kth/2D1432/2003/rltutorial.pdf.

Touzet, Claude F. 1999, Neural Networks and Q-Learning for Robotics. [document on-line] Consulted in 3/9/2003 Available in http://avalon.epm.ornl.gov/~touzetc/Publi/Touzet_IJCNN_Tut.pdf.

Zou Yi, Ho Yeong Khing, Chua Chin Seng and Zhou Xiao Wei 2003, Evidence method in ultrasonic sensor coalition for mobile robots. [document on-line] Consulted in 10/8/2003 Available in http://www.duke.edu/~yz21/research/YZ_IASTED-MIC2000.pdf.