Learning and Applying Artificial Intelligence with Mobile Robots

Julio César Sandria Reynoso[1,2], Mario Morales García[2], Arturo Mendoza Rendón[2]

[1] Instituto de Ecología, A.C., Departamento de Informática,
Xalapa, Veracruz, 91070, México
sandriaj@ecologia.edu.mx

[2] Universidad de Xalapa, Ingeniería en Sistemas de Cómputo Administrativo,
Xalapa, Veracruz, 91190, México

Abstract. Learning artificial intelligence sometimes is a very frustrating task when you try to understand it with a lot of theory and a little practice. This paper intends to illustrate that learning artificial intelligence could be an attractive and entertainment task when your try to build robots that show some intelligent behavior like see, hear, speech, move, and even learn. We use Lego mobile robots and Java for learning and applying some techniques from artificial intelligence like neural networks, genetic algorithms, computer vision, speech recognition, and speech synthesis.

1 Introduction

The use of physical robots as a teaching tool has been extended around the world. With the development of the Lego Mindstorms Robotics Invention System, some universities use them for educational purposes. The Robotics Academy in the Carnegie Mellon University (www.rec.ri.cmu.edu/education) offers a complete teaching package for children, to introduce them to technology, hardware, electronic control, computer programming and mechanics. It has been used to instill engineering skills, scientific interest, computer acquisition, general ideas and creativity among students [6]; also, applying Piaget's theories of cognitive development, to help students to understand concepts about complex dynamic systems, like how global behavior can emerge from local dynamics [8].

In the artificial intelligence arena, Lego based robots have been used for teaching neural networks [4] and for building low cost robotics laboratories for teaching artificial intelligence [5].

In this paper we use Lego mobile robots for teaching/learning some artificial intelligence themes in an Artificial Intelligence Course. Based on Nilsson's book [9], which covers these themes with software agents, we begin to work a neural network, a genetic algorithm and a computer vision algorithm with physical agents – our mobile robots.

With the idea of using only one programming language, available to Lego robots, we use Java. This way, we can build intelligent Lego robots with Java, as it provides APIs for programming systems that can see, hear, speak [7], and even learn [12, 13].

We use the Lego Mindstorms Robotics Invention System 2.0 for building the Lego mobile robots; leJOS 2.1.0, a little Java operating system for downloading and running Java programs inside the robots; Java 2 for compiling the Java programs under LeJOS, and some APIs for computer vision and speech recognition.

1.1 Lego Mobile Robots

The Lego Mindstorms Robotics Invention System (RIS) is a kit for building and programming Lego robots. It has 718 Lego bricks including two motors, two touch sensors, one light sensor, an infrared tower, and a robot brain called the RCX.

The RCX is a large brick that contains a microcontroller and an infrared port. You can attach the kit's two motors (as well as a third motor) and three sensors by snapping wire bricks on the RCX. The infrared port allows the RCX to communicate with your desktop computer through the infrared tower.

In this work, we use a Roverbot as it is constructed in the Lego Mindstorms Constructopedia, the guide for constructing Lego robots.

1.2 Java Technology

We use Java 2 Technology to program all algorithms we show, using own code and predefined packages and Java APIs as shown in Table 1.

Table 1. Java Technology used with Lego robots

Java Technology	Mean and use
J2SDK	**Java 2 Software Development Kit** to compile and run Java programs
LeJOS	**Lego Java Operating System** to run Java programs inside the RCX
LMbpn	A Lego Mindstorms Backpropagation **Neural Network** (own)
LMsga	A Lego Mindstorms Simple **Genetic Algorithm** (own)
JFM	Java Media Framework for **Computer Vision**
JSAPI & Sphinx	Java Speech API and Sphinx for **Speech Recognition**
JSAPI & Sphinx	Java Speech API and Sphinx for **Speech Synthesis**

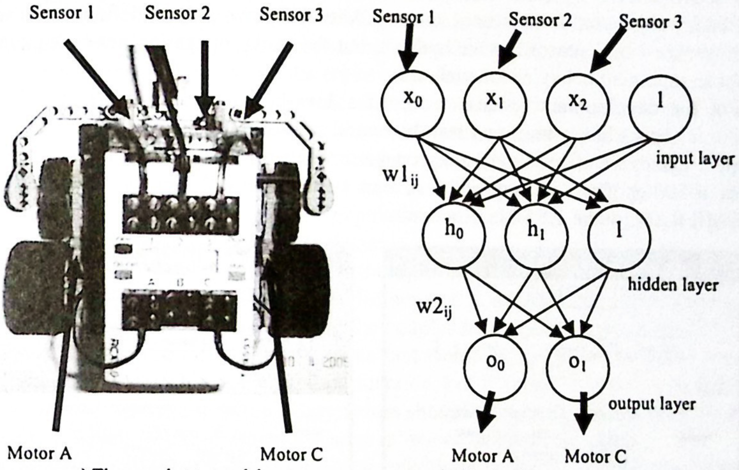
1.3 LeJOS

LeJOS is a small Java-based operating system for the Lego Mindstorms RCX. Because the RCX contains just 32 KB of RAM, only a small subset of the Java Virtual Machine and APIs can be implemented on the RCX. LeJOS can be downloaded from lejos.sourcefoge.net. For setting up a LeJOS installation refer to the LeJOS documentation or Ferrari *et. al.* [2].

2 Artificial Intelligence applications

2.1 Neural Network: The Backpropagation algorithm

For the neural network example, the task was to learn and apply the backpropagation algorithm as it is described by Rich and Knight [11]. We had to model a backpropagation network for our mobile robot. The robot has three inputs (two touch sensors and one light sensor) and two outputs (two motors) as shown in Fig. 1.a. So, we can use a three layer backpropagation network as shown in Fig. 1.b.



a) The roverbot to model b) The backpropagation network

Fig. 1. The robot input/outputs and its backpropagation network

The use of a three layered network and three units in the hidden layer was just an arbitrary decision influenced by teaching purposes.

To define input-output vector pairs for using in the backpropagation network, from the robot input-output (sensor-motor), we had to identify what was going to learn the robot. We defined four basic behavior rules:

1. **Forward**: if sensor 1 is off, and sensor 2 is over white floor, and sensor 3 is off, then Motor A and Motor C go forward (the robot goes forward).
2. **Turn right**: if sensor 1 is on, then Motor A goes forward, and Motor C goes backward (the robot turns right).
3. **Turn left**: if sensor 3 is on, then Motor A goes backward, and Motor C goes forward (the robot turns left).
4. **Backward**: if sensor 2 is over black floor, then Motor A and Motor C go backward (the robot goes backward).

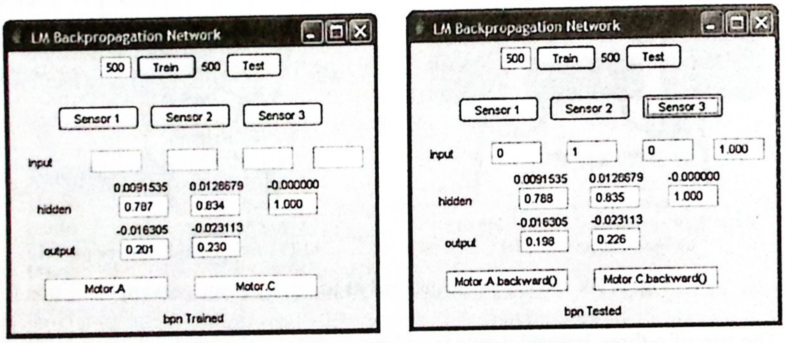
We translated these rules to training examples for the backpropagation network as shown in Table 2.

Table 2. The rules to learn as training examples

Rule	Training examples				
	Sensor 1	Sensor 2	Sensor 3	Motor A	Motor C
Forward	0	0	0	1	1
Turn right	1	0	0	1	0
Turn left	0	0	1	0	1
Backward	0	1	0	0	0
	Input vectors			Output vectors	

The input-output vector pairs are the examples we used to train the backpropagation network. This way, our mobile robot learnt to move forward, turn right, turn left and backward, based in its sensor states. Additionally, we did not define a rule when both, sensor 1 and sensor 2 were both on, but the backpropagation network gave the robot an emergent behavior for such case.

For the development process we used a Java-based graphic interface (Fig. 2), where it is possible to train and test the neural network. This way, at the final development cycle, we trained the backpropagation network, and found that it learnt the rules in 500 epochs, that required less than 1 second in a Windows XP system with 256 MB RAM, and a 1.8 GHz processor.



0.0091535 0.0128679 -0.000000
0.787 0.834 1.000
-0.016305 -0.023113
0.201 0.230

0 1 0 1.000
0.0091535 0.0128679 -0.000000
0.788 0.835 1.000
-0.016305 -0.023113
0.198 0.226

a) BPN training phase b) BPN testing phase

Fig. 2. A Java-based graphic interface for the development process.

The program code was compiled with the LeJOS compiler (`lejosc`) and downloaded to the RCX with the LeJOS loader (`lejos`), as shown in [12]. The program ran well in the RCX, but requires about 5 minutes to train 500 epochs. It is better to train the network in a desktop computer and then download the program to the RCX.

2.2 Genetic Algorithm

In order to give the mobile robot an artificial intelligence program that could by itself choose the correct way to response to the signals being received by its sensors an Artificial Neural Network application was created, but of course the program needs to be trained first, which poses a slight computational problem, the RCX by itself takes a considerable amount of time to process the information needed to be correctly trained, in this case when using the backpropagation algorithm took about 500 epochs to get the responses right, which translates to 5 minutes of processing time at the RCX. And, to learn an obvious emergent behavior – not defined by given rules in Table 2 – like both sensor 1 on and sensor 3 on, it took to the backpropagation network more than 5000 epochs to learn move backward.

The way chosen to tackle this problem was to optimize the backpropagation algorithm with the use of a simple genetic algorithm as described by Goldberg [3] immersed in the error minimization function, and get all the processing done on a desktop computer. This way, what was tried to accomplish was to get the correct weight values for each of the layers on the neural network, and then transfer those values to the neural network on the RCX, and so completing the training of the network without having to wait for the brick to get the entire math done.

The first thing done to get the genetic algorithm working was to create some useful functions that would be needed along the whole process more than one time. These were the functions to convert from real numbers to binary ones; the function to obtain the current values if the population when applied to the mathematical evaluation function, the roulette count for each of the population values and of course the function to convert back the binary numbers to real ones.

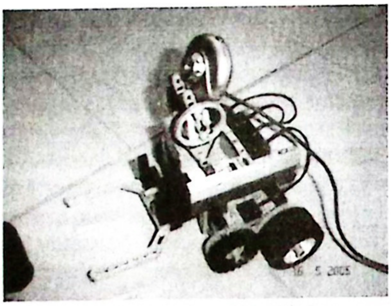
The next functions to get done where the most important ones, the coupling function and the mutation function. For the former one a few randomization operations were needed, especially when choosing the couples to reproduce and the point of insertion of each portion of the binary value that would finally create the new values. The latter one uses also a randomization function, but it's used to create a probability operation to choose if any of the binary values should mutate when passing to the next generation which would represent a change of one of its binary numbers.

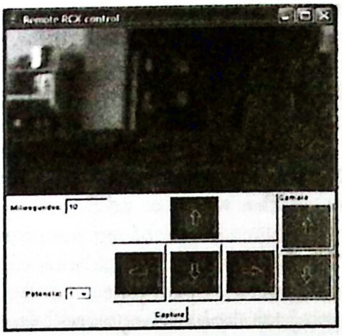
After getting all these functions done the only thing needed is to create an encapsulation for them, that is a program that uses the functions in the correct order and that using recursive programming can evolve the evaluation of the mathematical function in order to get the correct values of the weights by passing the values obtained after the mutation and coupling functions back to the genetic algorithm so it can start processing them all over again.

2.3 The Computer Vision Algorithm

One of the biggest issues in artificial intelligence is to simulate the way the human vision works, the computational power needed to correctly interpret the images captured in real time represents a huge obstacle to the development of this area. DeSouza and Kak survey the developments of robot vision navigation since 1980s [1]. Here, we applied just only one algorithm, mainly trying to discern different objects viewed across a camera. The whole point of it was to get the mobile robot to recognize and differentiate between two different objects and advance towards them in order to grab the one the user has ordered to be captured. This was planned to be done by using edge detection and remote controlling the robot.

Before even attempting to program anything the correct tools must be present, and one of these tools was the Java Media Framework [7] which presents libraries and methods to properly use the images captured by a camera, in this case a small web-cam attached to the robot's body (Fig. 3.a). Other tools are the LeJOS Vision classes in `vision.jar` and remote control classes `rcxcomm.jar` and `pcrcxcomm.jar` to remotely control the Lego robot through the infrared tower, manually using buttons in an own graphic interface (Fig. 3.b) or automatically using commands in the vision program.


a) A small webcam attached to the robot's body



b) The Java-based graphic interface to control the robot

Fig. 3. A small webcam for the robot vision

For this program one of the most basic edge recognition algorithms was used –the Sobel algorithm [10]– which uses convolution masks to determine the difference between pixels that are next to each other and find the edges on an image.

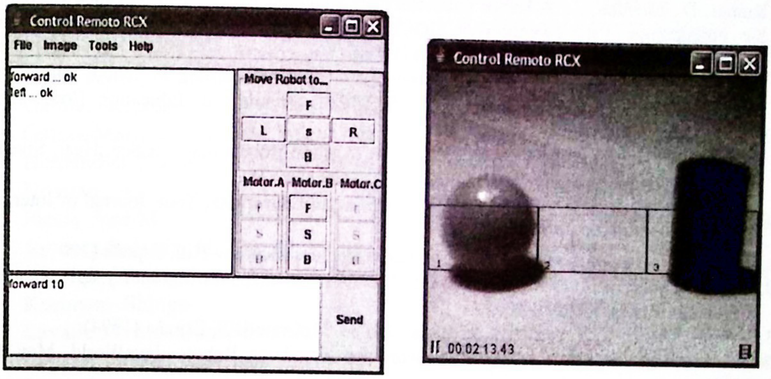
The biggest issue present when programming this application was to manipulate correctly the images by it's pixels. Therefore a function that can convert an image to an array of pixels on integer values it's needed. Fortunately Java provides a very easy to use method on its image libraries that is called PixelGrabber and it does precisely the work that is needed to be done. With the pixels already on an integer array it is only needed then to translate the Sobel algorithm into a function which at the end returns an array of pixels where the edges found get a different color. Another useful Java method can render an image out of the pixel array finally showing on screen the original image with the edges marked.

In this case we used two different objects to work with, a cylinder and a sphere. After getting the Sobel function working the next thing needed is to create a function that searches for the edge pixels and checks if they all make a geometrical figure a rectangle or a circle. The search is done by searching for pixels next to each other. To find the rectangle if two parallel lines are found connected to two perpendicular ones the image is recognized. For the sphere the pixels need to be connected in one of four ways - to the left and below, right and below, left and above and right and above – so the circle is drawn.

After that the work becomes simpler, a small program that lets the user choose between the object to look for needs to be done and using the Java Media Framework the image being captured it's shown to the user so he can check how the mobile robot looks for the geometrical figure on it.

2.4 Speech Recognition and Synthesis

In order to "talk" with our mobile robot a chat interface was designed to give orders to the robot and to receive response from it (Fig. 4.a). This way, we can see what the robot see (Fig. 4.b), and tell what to do, e.g., move forward, backward, turn right, turn left, move camera up or down, and so on.



```
forward ... ok
left ... ok
```

```
forward 10
```

a) A remote control chat interface b) The robot vision

Fig. 4. The remote control interface

Trying to give simple voice orders to the robot and receive responses from it in the chat interface, we used Sphinx 4, a Java Framework for speech recognition [14]. For the speech recognition we used a little grammar like is required by Sphinx:

```
grammar move;

public <greet> = ( go | turn | camera ) ( up | down |
left | right | stop | backward | forward );
```

3 Conclusion

Learning artificial intelligence in a one semester course using mobile robots is a more different and entertainment task than the traditional way – with a lot of theory and some programs. At the beginning students become disconcerted when the teacher tells them they have to learn and apply some artificial intelligence algorithms in a little Lego toy. But, it becomes very interesting to understand first how to program the Lego robot, and second, trying to program some algorithms, using only the Java language.

References

1. DeSouza, G.N., Kak, A.C.: Vision for Mobile Robot Navigation: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence (2002) 24:2, 237-267.
2. Ferrari, G., Gombos, A., Hilmer, S., Stuber, J., Porter, M., Waldinger, J. Laverde, D.: Programming Lego Mindstorms with Java. Syngress Publishing, Rockland MA (2002).
3. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989).
4. Imberman, S.P.: Theaching Neural Networks Using Lego-Handyboard Robots in an Artificial Intelligence Course. SIGCSE Technical Symposium on Computer Education (2003).
5. Kumar, D., Meeden, L.: A Robot Laboratory for Teaching Artificial Intelligence Resource Kit. Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education, Daniel Joyce (ed.), ACM Press (1998).
6. Law, K.W., Tan, H.K., Erwin, B.T., Petrovic, P.: Creative Learning in School with Lego Programmable Robotics Products. 29th ASEE/IEEE Frontiers in Education Conference (1999).
7. Meloan, S.: Futurama: Using Java Technology to Build Robots that can See, Hear, Speak, and Move. Sun Microsystems Inc, July (2003).
8. Miglino, O., Lund, H.H., Cardaci, M.: Robotics as an Educational Tool. Journal of Interactive Learning Research (1999) 10:1, 25-48.
9. Nilsson, N.J.: Inteligencia Artificial. Una nueva síntesis. McGraw-Hill, España (2001).
10. Parajes, G., De la Cruz, J.M.: Visión por Computador, imágenes digitales y aplicaciones. Alfaomega-RaMa, México (2002).
11. Rich, E., Knight, K.: Inteligencia Artificial. 2da ed. McGraw-Hill, España (1994).
12. Sandria Reynoso, J.C.: A Neural Network for Java Lego Robots. JavaWorld. May 16 (2005).
13. van Dam, B.: Simple Neural Network as robot brain. generation5, www.generation5.org (1999).
14. Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., Woelfer, J.: Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Sun Microsystems Inc. (2004).