2009-01-01

Autonomous Vehicle Navigation: A Comparative Study Of Classical Logic And Neural Network Technique

Javier Alejandro Flores
University of Texas at El Paso, jaflores@miners.utep.edu

AUTONOMOUS VEHICLE NAVIGATION: A COMPARATIVE STUDY OF CLASSICAL LOGIC AND NEURAL NETWORK TECHNIQUE

JAVIER ALEJANDRO FLORES

Department of Electrical and Computer Engineering

APPROVED:

Patricia Nava, Ph.D., Chair

Sarkodie-Gyan Thompson, Ph.D.

Rodrigo Romero, Ph.D.

Patricia Witherspoon, Ph.D.
Dean of the Graduate School

Dedication

To my parents Javier Flores and Virginia Garcia for all their unconditional support and guidance throughout my life; without it, I would not be the person that I am right now. Thank you so much. Los adoro.

AUTONOMOUS VEHICLE NAVIGATION: A COMPARATIVE STUDY OF CLASSICAL LOGIC AND NEURAL NETWORK TECHNIQUE

by

JAVIER ALEJANDRO FLORES, B.S.E.E

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

August 2009

Acknowledgements

I would like to thank everyone that has put their trust in me. First of all, my father and mother who have always been with me regardless of anything. Thank you papá for all the insights you have given me since I was little. I will always have with me all the inspirational advises that I have been gathering through all my life and have helped me to be the kind of person I am right now. Thank you mamá for all the comprehension and love you have given me since I was born. Thank you both for being my wonderful parents. Thanks to my brother Eduardo Flores and Claudia Flores for always being there and demonstrated love and understanding as well as helping me through difficult times. I want to express my special gratitude to Blanca Ramirez for all the support she has given me, for being so comprehensive, for being with me no matter what, and all those long days she spent with me during the achievement of this thesis.

I also want to thank Dr. Patricia Nava who put a lot of patience and trust in my work; thank you so much for all your advice and expertise you have provided me throughout my academic career and be there when I needed you the most. Special thanks to Dr. Sarkodie-Gyan Thompson for always teaching me how to fight and go forward, attacking the monster while is little. My appreciation to Dr. Rodrigo Romero for being a member in my committee and gave his professional advice for the completion of this work. I want to thank Dr. Eric Macdonald for all the skills I have acquired since I met him, “muchas gracias doctor”. I want to express one of my greatest gratitude to Professor Tony Woo who has guided me both professionally and personally. Thank you so much Professor Woo for all the trust and confidence you have put in me; I have to say that you represent a role model to me. A sincere appreciation to Ralph Loya and his family; thank you so much my friend for being there every time.

I want to thank Sukie Quezada, Linda Romero and Lily Chaparro for their kindness and support in this journey and for providing me the administrative resources in every semester.

To my friends Aaron Rodriguez, Damian Valles, and Alberto Enriquez thank you so much for being with me and supported me. A special thanks to Carlos Leija, who has helped me tremendously and has been part of my life since we were in high school; thank you my friend.

Abstract

Today is an era in which humans need the help of automated machines to facilitate their hectic lives. Over the past few decades, robotics has been one of the most researched areas in the world. One area of this research is obstacle avoidance for autonomous vehicles. In order to avoid an obstacle, the system may include two important characteristics: obstacle detection and avoidance control. In this thesis a classical logic system and artificial neural network approach is presented. With the help of an Integrated Development Environment, a virtual robot was able to negotiate a maze and avoid obstacles according to the data gathered from its sensors. The latter approach was compared to an Artificial Neural Network (ANN) configuration, which proved to perform with successful results. This thesis demonstrates that the use of classical logic systems and ANN offers a good solution to the problem of obstacle-avoidance while negotiating a maze environment. ANN can be considered to be faster, once the neural network is trained, in response to obstacle-avoidance due to its massive parallel processing.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Intelligent Systems (IS) is one of the most researched areas around the globe. With the knowledge gained in the area of Artificial Intelligence (AI) during the early years, IS has expanded beyond AI to include many diverse techniques that seek to emulate human behavior. In many years, scientists and engineers have come together and applied all their knowledge and skills to machines in order to get their function closer and closer to human reasoning. Machines, more and more, are being introduced into factories, where in fact they are replacing human workers. Governments, all over the world, invest millions of dollars on robotic research. One of the main reasons for this focus and ambition is because of machine efficiency and consistency. A robot never tires or loses interest in its tasks. It can basically go on performing twenty-four hours a day, if necessary, with only an occasional weekly or monthly maintenance session, and it will do all its tasks without boredom or fatigue.

Robotics engineers are very interested in emulating the sensing of organs that human beings own, as closely as possible. For example, one of the most important human senses is touch, which produces a variety of distinct sensations including cold, heat, pain and pressure. In fact, touch is likely to be the most important aspect in a robot.

In order to duplicate, in some way, the behavior of humans and animals, a clever technique was born in the 1940s where the first neural network computing model was introduced by McCulloch and Pitts. Another technique implemented in machine control can be described as “Classic Logical” control, in which a series of statements are performed sequentially producing an outcome.

In this thesis, a comparison of Artificial Neural Networks and Classic Logical Control is presented, and their performance in controlling an Autonomous Vehicle is examined.

1.1 THESIS ORGANIZATION

This thesis has the following structure: Chapter two will address the foundations of Artificial Neural Networks (ANN) as well as related work; it will include ANN basics, such as structure and functionality, as well as a discussion of ANN manipulations with respect to training concepts. Chapter three will focus on Classical Logic Control and its purpose in this thesis. In addition, it will give an introduction to the problem at hand in order to provide context for the remaining sections of this thesis. Chapter four will present the comparison of the two techniques mentioned above, as well as provide a detailed explanation of results, according to the different methodologies, in achieving control of an autonomous vehicle. Finally, chapter five will summarize the outcome of the testing results and will give a brief discussion and suggestions about future work involving an implementation of a physical autonomous vehicle.

Chapter 2: Artificial Neural Networks: Background and Related Work

2.1 BIOLOGICAL NEURAL NETWORKS

Study of the human brain spans many centuries, from examination of human behavior to dissection to modeling and simulation. Researchers have studied biological neural networks (BNN) for many years; development of a mathematical model is considered the starting point of study for Artificial Neural Networks (ANN). The illusion of imitating the human brain began in trying to make a machine "reason" and "decide" in the same manner. The brain has an incredible gift of learning; it processes everything rapidly, e.g. it can be trained to recognize complete and incomplete patterns. Biological trained networks continue performing even if certain neurons fail. For example, in a crowded noisy environment, a person can recognize someone, from a distance, as well as distinguish voices [Kar96]. Generally speaking, learning is the process by which the neural network adapts itself to stimuli and eventually, after proper adjustment, produces a desired response.

Inside a human brain, and in many other organisms, there exist millions and millions of cells called neurons. Figure 2.1 illustrates an example of a single neuron.

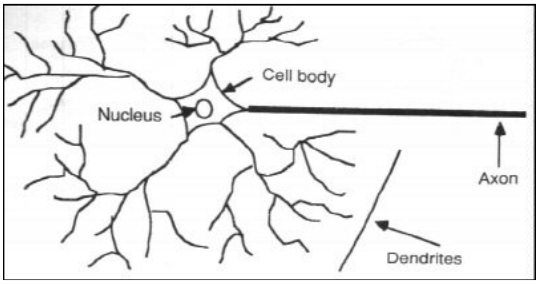

Figure 2.1. Representation of a neuron.

A biological neuron has three main components that are of particular interest: dendrites, axon, and soma (cell body). Dendrites are fine structures that help interconnect neurons with each other. These

interconnections are paths where electrical signals (impulses) are sent across a synaptic gap by means of chemical processes, which modify the incoming signal by scaling the frequency of it. The soma (or cell body) adds the incoming signal, and when sufficient input is received, the cell fires; it transmits a signal over its axon to other cells. The human brain has been found to have more than 3×10^{11} neurons where each neuron can fire about 100 times per second; therefore, the brain can possibly execute 1×10^{3} to 1×10^{4} synaptic activities per second [Sch97, Kar96]. In addition, research suggests that more than 50 properties in the configuration of neuronal interconnections affect the information transmitted. Considering that a received sensory pattern via the synapses probably excites a relatively small percentage of sites, it is proven that an almost endless number of patterns can be presented at the neuron without saturating the neuron's capacity [Kar96]. Two types of synapses occur after a neuron fires: excitatory and inhibitory. It is believed that the resultant action can be computed from the algebraic sum of both activities, excitatory and inhibitory, where the resultant action will be performed if the combination of the excitatory and inhibitory actions exceeds the threshold characteristics of the cell.

A basic description of biological neurons' operation can be summarized as follows: neurons receive the input from other neurons by means of the synapse, which happens to be the physical connection between a dendrite and a neuron. According to Zurada, "interneuron communication is sometimes electrical but is usually affected by release of chemical transmitter in the synapse. The neuron is able to respond to the total of its inputs aggregated within a short time interval called *period of latent summation*. Response is generated (through the axon) if the total potential of its membrane reaches a certain level (threshold)" [Zur92].

Biological neurons can be thought of as millions of tiny data processors that work together to solve a particular problem. Individual neurons are much slower than an old computer, but the human brain is still capable of easily executing certain complex tasks that any other powerful machine either cannot complete, or take much longer to complete. Some examples of these tasks include understanding

human language and identifying objects by sight, sound, smell, touch, and taste. Other examples include the ability to feel and respond to emotions and the ability to physically express these emotions through poetry, music and art. According to Ronald Kotulak, “when it comes to building the human brain, nature supplies the construction materials and nurture serves as the architect that puts them together” [Kot97].

There is a huge diversity in biological neurons even though they possess the same set of genes; e.g., individual neurons can activate only a small subset of neurons, such as is the case in the visual or auditory system. This diversity adds to the complexity of the neural network. In summary, there are certain properties that all neural networks possess,

- Many parallel connections exist between many neurons.
- Connections provide feedback to other neurons and to themselves.
- Neurons excite some neurons, while inhibiting the operation of others.
- Neurons are capable of being synchronous in operation.

2.2 WHAT IS AN ARTIFICIAL NEURAL NETWORK (ANN)?

An ANN is classified as an "imitation" of neurons, such as those congregated inside either a human or animal brain. These neurons work in unison in order to solve complex problems. One characteristic that makes an ANN so special is that it can learn by examples, as human beings do. An ANN can be configured in many ways depending on the proposed application, e.g. pattern recognition or data classification. ANNs have become a useful tool since the idea was conceived, with paradigms created by the pioneer work of several researchers, making a rapid transition from the cognitive and neurobiology field to engineering [Kar96]. From here on, the designation ANN and NN will be used interchangeably, since the neural networks discussed in this work are not biological, which are typically abbreviated BNN. Artificial neurons work by example and by training, just like biological neurons. Artificial neurons have to tweak themselves in order to have the right output, and this is called "training." In addition, this so-called "training" can be proven effective, mathematically. That is to say, it can be mathematically proven that if a solution exists, then the training process will find the solution (i.e. a set of weights) in a finite number of iterations.

ANN is classified in three ways: (1) its pattern of connections called "architecture," (2) its method of determining the weights on its connections called "training," and (3) its activation function. According to [Fau94], a neural net (NN) consists of a large number of simple processing elements called *neurons, units, cells,* or *nodes.* Each neuron is connected to other neurons by means of directed communication links, each with an associated weight that is used by the net to solve a problem. Figure 2 illustrates a simple artificial neuron, where inputs (X_1, X_2, and X_n), weights (w_1, w_2, and w_n) and activation function can be observed.

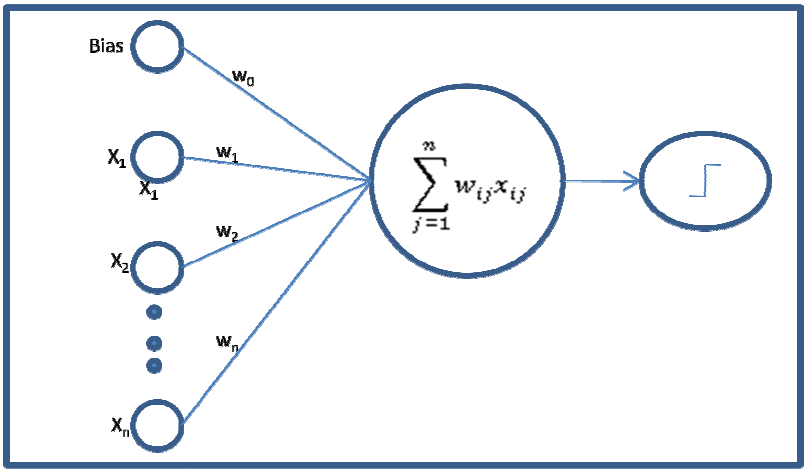

Figure 2.2. A simple (artificial) neuron.

As mentioned before, a NN processes information but not with a sequential algorithm. Meaning that this process is based on parallel decomposition of complex information into basic elements; i.e., a NN decomposes complex information into its fundamental elements, and these elements and their relationship to each other are stored in the brain's memory banks [Kar96].

In order to explain how a neuron works, a mathematical model describing the biological system's functionality is needed. Moreover, a computer can be utilized to perform and simulate the model's algorithm and finally get a close description of the biological system: an algorithm that can be modified to either enhance the systems' performance or to simplify it.

Taking into consideration the latter, several key features that ANN offers are suggested directly from BNN, such as [Fau94]

- A synapse's strength may be modified by experience
- Signals may be modified by a weight at the receiving synapse
- The processing element sums the weighted inputs
- The neuron transmits a single output
- The output from a particular neuron may go to many other neurons

- Memory is distributed
    - Long-term memory resides in the neurons' synapse or weights
    - Short-term memory corresponds to the signals sent by the neurons

2.3 BASIC MODEL OF A NEURON

Neurons possess an internal state, called *activation*, which is a function of the inputs it receives. Typically, a neuron sends an activation signal to several other neurons. When a neuron is part of a network of many other neurons, it is referred to as a node. It is important to remember that a neuron can send only one signal at a time although that signal broadcasts to several other neurons [Fau94].

For example, consider again the example in figure 2.2 where a basic model of a neuron is depicted. The neuron has a set of n inputs x_j, where the subscript j takes values from 1 to n and indicates the source of the input signal. Each input x_j is weighted by a factor of w_j i.e. x_j is multiplied by w_j. In addition, the neuron has a bias term w_0, a threshold value, which has to be reached or exceeded in order to produce a signal, an activation signal, and an output. It is important to note that the output of a neuron can be an input to other neurons [Kar96]. Inputs, weights, activation signals, outputs, threshold, and nonlinear function are written as x_{ij}, w_{ij}, R_i, O_i, Θ_i, F_i, respectively.

The transfer function of the basic model is described by the relation

$$O_i = F_i\left(\sum_{j=1}^{n} w_{ij}x_{ij}\right),$$

and the neuron's firing condition is defined as

$$\sum_{j=1}^{n} w_{ij}x_{ij} \geq \Theta_i \,,$$

where the index i represents the neuron in question and j represents the inputs from other neurons. The purpose of having a non-linear function is to ensure that the neuron's response is bounded, i.e. a conditioned output, as a result of large or small activating stimuli and thus is controllable [Kar96].

Figure 2.3 illustrates the various types of nonlinearity functions used depending on the paradigm at hand; the most common are the sigmoid and the hard limiter.

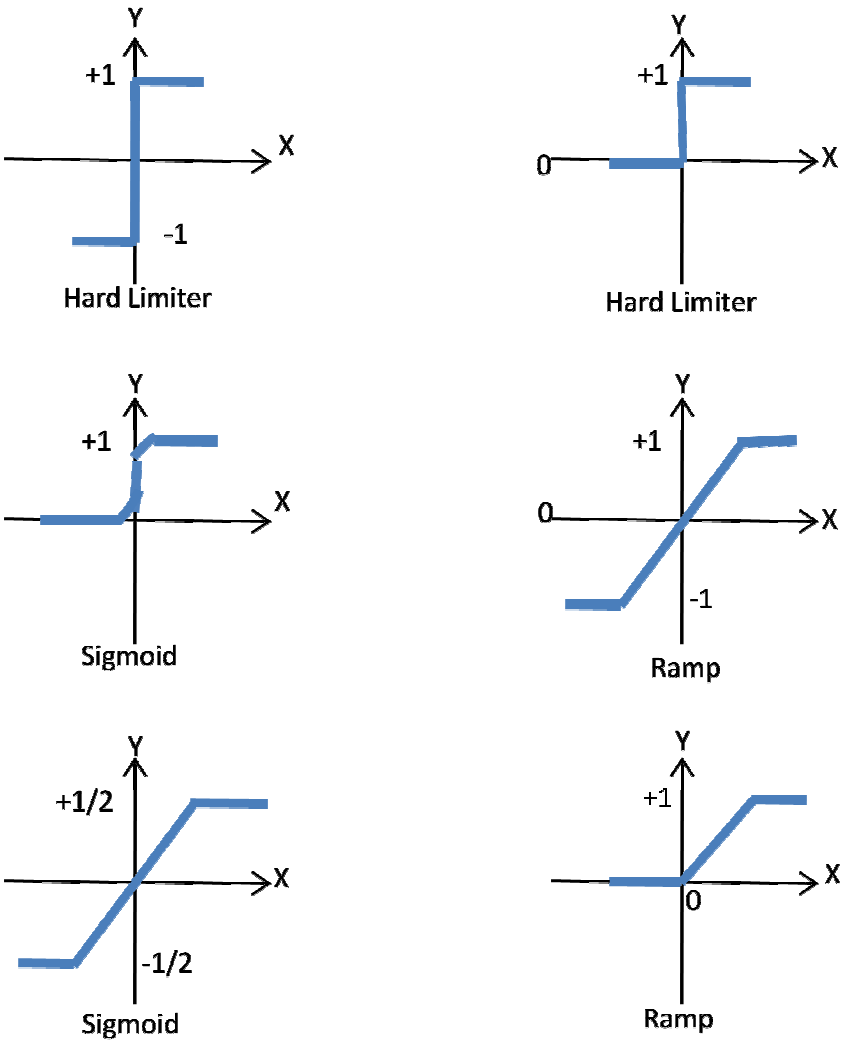

Figure 2.3 Types of Nonlinearity Functions

2.4 NEURAL NETWORK CLASSIFICATION AND TOPOLOGIES

The process of learning is primarily developed in two ways: supervised and unsupervised. The name of supervised learning comes from a method where the response of the net is compared with a desired output signal, referred to as target response. If the actual response differs from the target response, the NN generates an error, which is then used to calculate adjustments that should be made to the network's synaptic weights so that the actual output matches the target output [Kar96]. In other words, the error is minimized, if possible to zero, depending on the algorithm used.

In contrast, there is the unsupervised learning where the algorithm does not have a target output. In this type of learning the NN receives many different excitations, or input patterns, and it arbitrarily organizes the patterns into categories. When a stimulus is later applied, the NN provides an output response indicating the category to which the stimulus belongs.

There are a many parameters that a NN designer should take into consideration; such parameters will help the net to be more reliable [Kar96]

- Network topology
- Number of layers in the network
- Number of neurons or nodes per layer
- Learning algorithm to be adopted (in either the supervised or unsupervised case)
- Number of iterations per pattern during training
- Number of calculations per iteration
- Speed to recall a pattern
- Bias terms
- Threshold terms (occasionally set to some a priori fixed value, such as 0 or 1)
- Boundaries on the synaptic weights

- Choice of nonlinearity function and the range of operation of the neuron

ANNs are interconnected in a certain way depending on the job for which they are designed. Based on these topologies, the layer where the input patterns are applied is the **input layer**. The layer from which the output response is obtained is the **output layer**. Intermediate layers are called **hidden layers** because their outputs are not visible [Kar96, Sam07]. Some of the classical topologies are listed in figure 2.4, which basically depicts single-layer and multi-layer networks.

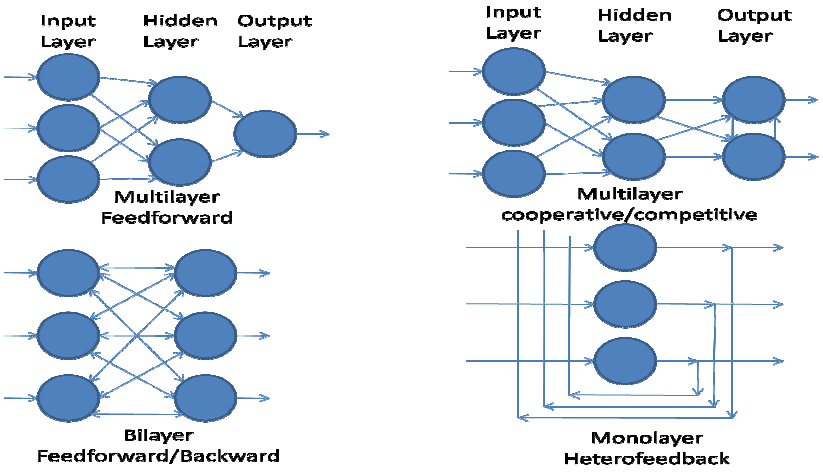

Figure 2.4 Topologies in neural networks

2.5 NEURAL NETWORK PARADIGMS

Paradigms are present in everyday life. They have to serve as a model for the neuron so the neurons can learn and react accordingly. In most network topologies, the design of their corresponding training algorithms is essential. In order to have a successful network, an effective and simple enough training algorithm must be created [Cho07].

Figure 2.5 illustrates an artificial neural network in which neurons are denoted as circles, and arrows represent the communication paths between neurons that at the same time include the synaptic strengths among them.

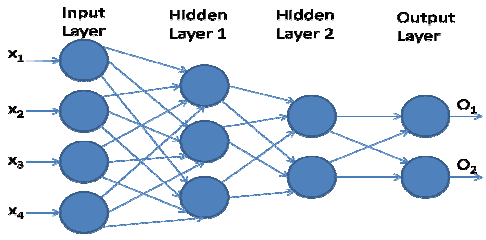

Figure 2.5 Feedforward Multilayer Neural Network

Generally speaking, a NN can be considered as a sophisticated signal processor. The processing procedure of this network is classified as **distributed,** and it is stored at the synapses of each neuron. During the process of learning, weights and thresholds are adjusted continuously until a desired output is reached. In other words, the strength of each synapse and the threshold value of each neuron constitute the network's program. This part of the chapter deals with paradigms that are essential for the understanding of ANN.

2.5.1 McCulloch Pitts Model (MCP)

In 1943, the neurophysiologist McCulloch and his associate Pitts employed an electrical open-loop circuit to model a simple single neuron. This model was the first ever created to serve as a foundation for subsequent developing paradigms [Sam07]. The MCP model is quite simple; no mechanism exists to compare the actual expected output response, and thus no weight adjustment and no learning takes place [Kar96].

In this model each input receives a stimulus x_j that is weighted by some value w_{ij} that represents the synaptic strength. All weighted inputs are summed in the neuron, and if the combined input reaches a certain threshold, a response is generated. A **nonlinear transfer function** f is generated, expressed by

$$O_i = f\left(\sum_{j=1}^{N}(x_i w_{ij}) - \Theta_i\right),$$

where x_{ij} is the incoming signal, or stimulus, at input j on the ith neuron, f is the nonlinearity, and O_i is the output response of the ith neuron. In this model the bias term Θ_i and the weights w_{ij} are assumed to have reached a steady state [Kar96, Sam07].

2.5.2 The Perceptron

Considering that the MCP model implements neither adaptation nor learning, the need for feedback was immediately suggested. This need for control and adaptability was introduced by Frank Rosenblatt in 1962. This model is considered the simplest type of NN and is typically used for classification in which the network simulates an associative memory. This paradigm can classify and recognize abstract and geometric patterns; i.e., it needs a supervised learning algorithm that will compare the error between the target and the actual output during the learning process. Figure 2.6 illustrates the perceptron model, which is very similar to MCP [Cho07].

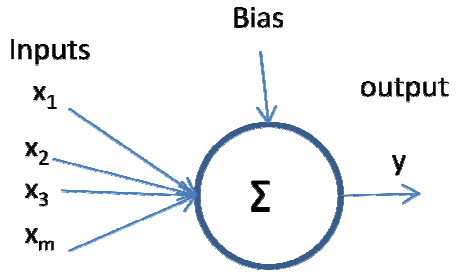

Figure 2.6 Single-layer perceptron model

The mathematical model for a perceptron is described as follows:

$$y_{in} = \sum_{i=0}^{n} (w_i x_i)$$

and

$$y = f(y_{in})$$

where

X_i= input signals, i…n

W_i= corresponding weight of X_i

f(·)= activation function

2.5.3 Multilayer Feed Forward Network Perceptron

Other kinds of perceptron models have been suggested since the invention of the original. In general, multi-layer Perceptron (MLP) nets are composed of many simple perceptrons in a hierarchical structure forming a feed-forward topology with one or more layers (**hidden layers**) between the input and the output layers, (see figure 2.7) [Kar96]. It is important to state that the number of hidden layers and the number of neurons per layer is not fixed; i.e., depending on the application, the net can have different number of layers and neurons interacting with each other.

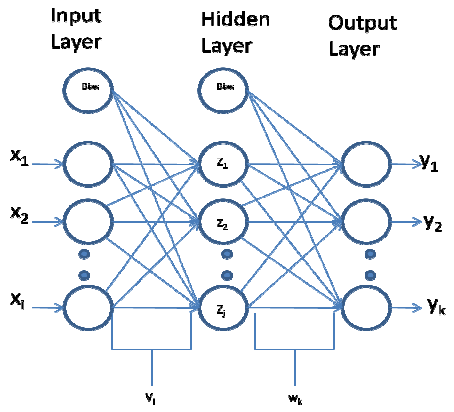

Figure 2.7 Layer definition in MLP

From figure 2.7, it is defined that the network is conformed of **i** input neurons, **j** hidden neuron and **k** output neurons. Having this in mind, the following is extracted: an input vector **X**, a hidden vector **Z**, and a target vector **T**. As for the matrices, a hidden layer weight matrix **V** and an output layer matrix **W**. The latter can be expressed like the following:

$$\mathbf{X}=[X_0, X_1,.., X_i],$$

$$\mathbf{Z}=[Z_0, Z_1,.., Z_j],$$

$$\mathbf{T}=[t_0,t_1,\ldots,t_k],$$

$$V = \begin{pmatrix} V_{01} & V_{02} & V_{0j} \\ V_{11} & V_{12} & V_{1j} \\ V_{i1} & V_{i2} & V_{ij} \end{pmatrix},$$

and

$$W = \begin{pmatrix} W_{01} & W_{02} & W_{0k} \\ W_{11} & W_{12} & W_{1k} \\ W_{j1} & W_{j2} & W_{jk} \end{pmatrix}$$

From the structure, it is assumed that **X, V** and **W** must be known or given, unless the network needs to be trained. In that case, only **X** and **T** are necessary, as **V** and **W** must be obtained through training. Furthermore, the values of the bias are fixed to 1.

Any structure composed of an input layer, one hidden layer, and one output layer can approximate any continuous multivariable function to any accuracy taking into consideration enough neurons in the hidden layer [Fau94]. For this reason, this type of architecture is often used as a model-free estimator.

2.6 TYPES OF TRAINING ALGORITHMS

Many types of training algorithms exist in the area of ANNs. There are some algorithms typically associated with certain ANN structures, and some algorithms are typically used for certain types of applications. So there are many algorithms, and specific reasons to choose one over another. While each has given benefits and drawbacks, this section will focus on two types of algorithms, Winner-takes-all and Back-Propagation.

2.6.1 Winner-takes-all

This algorithm is used in cases where competitive unsupervised learning is needed. Basically, the structure of the model will have a single layer of ***N*** nodes with their own set of weights w_n. Then, an input vector is applied to all nodes providing an output $O_n = \sum_j w_{nj} x_j$. The winner node is selected according to the best response based on the applied input [Kar96]. The criterion is the following:

$$O_n = \max_{i=1,2,\ldots N} (w_n x)$$

The change of weights is calculated according to

$$\Delta w_n = \propto (k)(x - w_n)$$

where is a small positive scalar known as the **learning rate**. This scalar may decrease or increase throughout the learning process and the winning weight vector is considered to be the closest to the input ***x***. Figure 2.8 illustrates a competitive learning architecture denoting node 1 as winner.

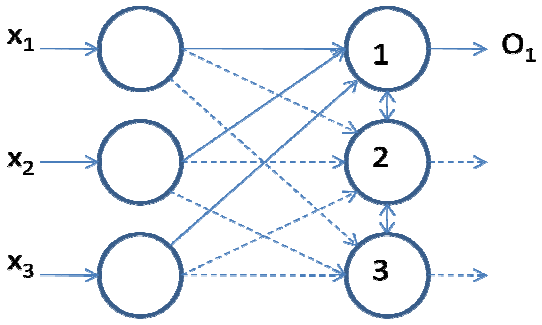

Figure 2.8 Competitive Learning with node 1 winning

2.6.2 Back Propagation

The back propagation (BP) algorithm was developed by Paul Werbos in his dissertation in the year 1974, wherein he describes the process of learning through back propagation of errors. BP has been widely used as a learning algorithm in MLPs. In essence, the algorithm learns a distributed associative map between the input and output layers.

As mentioned, it is difficult to adjust the weights in the hidden layers; the more hidden layers there are in a network, the more difficult it is to adjust their weights. The ultimate goal is to minimize the error in the difference between the actual and desired (target) outputs. The back propagation algorithm involves iterative processes based on its training equations; therefore, a computer or a microcontroller is used to alleviate this problem and have a more accurate result [Kar96, Fau94, Nau97]. The following characteristics are needed before starting a BP algorithm:

- A set of training patterns, input and target
- A value for the learning rate
- A criterion that terminates the algorithm
- A methodology for updating the weights
- A nonlinearity function (usually the sigmoid)
- A list of initial weight values (typically small random values)

The process starts in applying the first input pattern X_k and the corresponding target output T_k. The input causes a response to the neurons of the first layer, which in turn causes another response to the neurons of the next layer until an output is obtained at the output layer. This response is compared against the target response calculating the differences. The error is taken by the algorithm and used to compute the rate of change. Note that this process has been applied only in a forward fashion. Afterwards, the

algorithm steps back one layer before the output layer and it recalculates the weights of the (output layer) so the error is minimized. This process is followed until new weights, moving layer by layer backwards toward the input, are recalculated. Once the input is reached and the weights are calculated, the algorithm proceeds to select the next pair of input-target patterns, $[X_k, T_k]$, and repeats the process [Nau97]. The process is called back propagation because the responses move in a forward direction throughout the network, but the weights of the neurons are recalculated by moving backwards. One advantage of BP, in its current form, is that the algorithm adds a momentum; e.g., training with a too small learning rate produces a slow process, whereas a very large learning rate will proceed much faster but it can produce oscillations between relatively poor solutions. Thus, adding a momentum will avoid the latter problem by varying the magnitude of the change, i.e. magnitude of the negative gradient in the weight space, when approaching the minimal solution.

When applying an activation function, characteristics such as the following should be considered: it should be continuous, differentiable, and monotonically non-decreasing; in addition, its derivative should be easy to compute [Fau94]. One of the most typical activation functions is the binary sigmoid which has a range of (0, 1), (figure 2.9). This activation function is defined as

$$f(x) = \frac{1}{1 + \exp(-x)}$$

with its derivative as

$$f'(x) = f(x)[1 - f(x)]$$

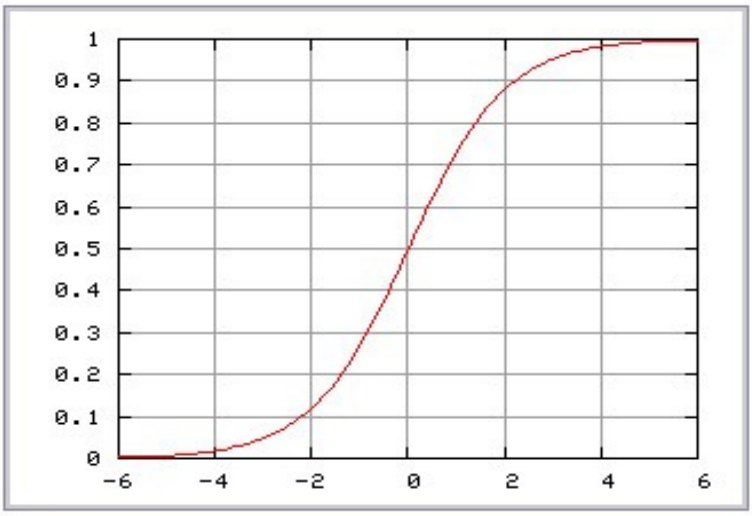

Figure 2.9 Binary Sigmoid

The problem with the binary sigmoid is that when learning, sometimes it does not converge properly. In order to alleviate the latter a binomial sigmoid is applied as follows. Figure 2.10 illustrates the latter function.

$$f(x) = \frac{2}{1+\exp(-x)} - 1,$$

with its derivative as

$$f'(x) = \frac{1}{2}[1 + f(x)][1 - f(x)]$$

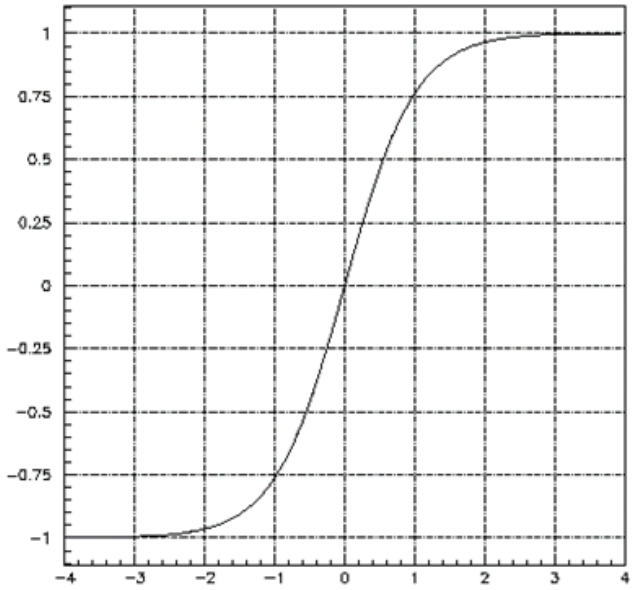

Figure 2.10. Binomial Sigmoid

According to Fausett, the algorithm for FF Back-propagation is described below. Note that the algorithm is divided into three stages: feed forward propagation of the input vector, error calculation and back propagation, and weight adjustments.

Step 0. Initialize weights. (set to small random values).

Step 1. While stopping condition is false, do Steps 2-9.

Step 2. For each training pair, Steps 3-8.

Feedforward:

Step 3: Each input unit (X_i, $i=1,\ldots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units)

Step 4: Each hidden unit (Z_j, j=1, …, p) sums its weighted input signals

$$z_{in_j} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

applies its activation function to compute its output signal

$$z_j = f(z\_in_j),$$

and sends the signal to all units in the layer above (output units).

Steps 5: Each output unit (Y_k, k = 1, …, m) sums its weighted input signal

$$y\_in_k = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y\_in_k)$$

Backpropagation error:

Step 6: Each output unit (Y_k, k=1, …, m) receives a target pattern corresponding to the input training pattern computes its error information term.

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

calculates its weight correction term (used to update w_{jk} later)

$$\Delta w_{jk} = \alpha \delta_k z_j$$

calculates its bias correction term (used to update w_{ok} later)

$$\Delta w_{ok} = \alpha \delta_k$$

and sends δ_k to units in the layer below.

Step 7: Each hidden unit (Z_j, j = 1, …, p) sums its delta inputs (from units in the layer above)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

multiplies by the derivative of its activation function to calculate its error information term

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and calculates its bias correction term (used to update v_{oj} later)

$$\Delta v_{oj} = \alpha \delta_j$$

Update weights and biases:

Step 8: Each output unit (Y_k, k = 1, …, m) updates its bias and weights (j=0, …, p):

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Each hidden unit (Z_j, j = 1, …, p) updates its bias and weights (i = 0, …, n) :

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

Step 9. Test stopping condition.

The original motivation for applying back-propagation to the net is because of its ability to achieve a balance between a target and the responses to the set of inputs i.e. a balance between memorization and generalization. In order to have accuracy in the generalization (in the algorithm previously mentioned), the stop condition is generally set to 0.05 (depending on the problem to be solved). The following error test is commonly used to stop a training process.

$$square\ error = \frac{(target - y\_actual)^2}{training\_set\_size}$$

Chapter 3: Classical Logic Control System

In the previous chapter, the concepts and different architectures for ANN were explained. This chapter will be dedicated to describing the area of "Classic Logical Control," its contrast to ANNs, and its effects on the application in this thesis. After discussing general concepts, an introduction to the approach of a given problem will be addressed.

3.1 LOGIC

In 1920, logic was mostly an indefinite vast area of research. Since the 1970s, this vast area has been clustering into fields of study for mathematics, computer science, artificial intelligence, and linguistics, just to mention a few. New areas in this field of logic have brought new important components such as imperative, declarative and functional programming; verification of programs; knowledge-based systems and so on [Sho97]. In humans, logic is nothing more than a small part of the capacity to reason. Logic can be a means to compel us to infer correct answers, but it cannot, by itself, be responsible for our creativity or for our ability to remember [Ros05]. Logic can help people in their ability to make sense, but not necessarily to enable us to remember what factors actually made sense. Logic, according to the Merriam-Webster dictionary, is defined as the "science that deals with the principles and criteria of validity of inference and demonstration: the science of the formal principles of reasoning." In other words, for humans, logic is a way to interpret things quantitatively: developing a reasoning process that in fact can be replicated and manipulated with mathematical precepts. The interest in logic is the study of truth in logical propositions; in classical logic, this truth is binary – a proposition is either true or false. [Ros05]

3.2 CLASSICAL LOGIC

In classical logic, a proposition is a statement that is contained within a universe of elements that are strictly constrained to be either true or false. The veracity of an element in the universe can be assigned to a binary truth value; in the Boolean system, the latter is interpreted as a value of 1 (true) or 0 (false).

There are five logical connectives that help to form logical expressions involving two simple propositions. Table 3.1 illustrates the logical connectives.

Table 3.1. Logical Connectives

Disjunction	OR	(/\)
Conjunction	AND	(\/)
Negation	NOT	(-)
Implication	IMPLIES	(→)
Equivalence	IF AND ONLY IF	(↔)

The logical OR, (/\), is commonly referred as “inclusive or.” In order to distinguish this logical OR from the natural language OR is that the latter implies exclusion e.g. when a person is in a restaurant and has the option of selecting “toast OR pancakes” it is understood that the option has to be one or the other, but not both. Whereas the “inclusive or” implies that either or both options can be selected.

In the case of equivalence, a fact of dual implication arises, e.g. in the case of two propositions P and Q we can have the following:

If $P \rightarrow Q$ and $Q \rightarrow P$, then $P \leftrightarrow Q$

Having this explanation in mind, one can deduce that from the five logical connectives, compound propositions can be formed i.e., one proposition can lead to many others at the same time.

The latter implication is also well known as a classical implication. In the example above, P is also referred to as the hypotheses or the antecedent, and the proposition Q is also referred to as the conclusion or the consequent. [Ros05]

The implication IF A, THEN B, (P→Q), ELSE C can be illustrated in a Cartesian Space with typical sets of A, B and C. The latter is illustrated in figure 3.1.

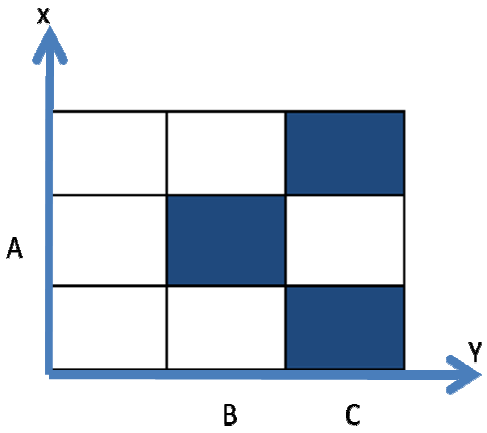

Figure 3.1. Cartesian Space with typical sets of A, B and C

As it was mentioned, classical logic statements can have only two possible truth values: either true or false. One way to represent such statements can be summarized in a truth table. This truth table, table 3.2, illustrates all possible truth value combinations and demonstrates the results for each logic connective.

Table 3.2. Truth table of Logic Connectives

P	Q		P∧Q	P∨Q	P→Q
False	False	True	False	False	True
False	True	True	False	True	True

True	False	False	False	True	False
True	True	False	True	True	True

Control system is an area in which classical logic is based upon. A control system can be defined as "the interconnection of components forming a system configuration that will provide a desired system response."[Dor98] The latter refers to what is intended in this thesis, which is an implementation of actions for an autonomous vehicle based on sensors' data. In order to analyze a system, a linear system theory must be applied that assumes a cause-effect relationship for the components of a system. Furthermore, a component of a system can be expressed as a block, where an input-output effect is represented. The latter is illustrated in figure 3.2.

Figure 3.2. Process to be controlled

The block depicted in figure 3.2 is essentially what a classical logic system will process and react accordingly with respect to its sensors; it will gather information from its sensors and will react accordingly to avoid collisions as well to attempt in finding a goal. In the following chapter will be explained how these processes will perform. The concepts of both classical logic and ANN will take place in the implementation of the autonomous vehicle. The scenario, where these two concepts are going to be applied on, will be a maze. This particular maze will be generated in a platform environment, in which a virtual robot can be simulated as well. Chapter four will give further details about these implementations and their results.

Chapter 4. Discussion of Problem, Approach and Results

As it was mentioned in chapter 3, the concept of classical logic control and ANN will be discussed in this section as well as how they would be applied to solve a given problem.

Robotics can be considered one of the fields that has currently experienced more exploration than ever before. There are robots everywhere in our hectic daily life, even in their most simplistic way, e.g. an automated car-wash, dishwashers, clothes dryers, etc. In fact there is no concrete universal definition of what a "real" robot is, but for this thesis a robot can be considered to have the following attributes:

- a collection of sensors;
- a program that defines the robot's behavior; and
- a collection of actuators and effectors

Now that a robot has been defined, for the purpose at hand, a scenario for this robot must be defined as well. The selected scenario is probably the most popular "problem" that people like to tackle in their free time, since time immemorial: a maze. Mazes exist virtually everywhere: city streets, building hallways, highways, puzzle books, electronic games, etc. All of them are alike in one thing, in that the individual starts at one point, and needs to traverse the maze to find the exit, which is at another point. The next section will give details on the dimensions of the maze for this study, as well as specifications of the robot used.

4.1 ROBOT SPECIFICATIONS

The usage of an Integrated Development Environment (IDE) is necessary in this thesis in order to have a user-friendly framework that enables to program in many modern programming languages, and furthermore simulate a robot. An IDE typically consists of a code editor, a debugger and a graphical user interface. It is important to mention the importance of simulating a robot way before manufacturing a real one. A real robot, many times involves a lot of decisions to be made such as calibration of sensors, mounting and remounting sensors while going through numerous programming alternations and tests to see how the robot would react to the different instructions. The main reason of using a simulator is to avoid the latter, and all the alterations that affecting the robot can be done in a fraction of a time. Having said the previous aspects, the IDE used in this thesis has a collection of sensors that enables a virtual robot to see and feel its environment. The virtual robot is equipped with the following sensors:

- five Infrared sensors;
- one camera;
- four bumper sensors;
- line detector sensors;
- one ranging sensor;
- beacon detection;
- compass; and
- Global Positioning System.

In this thesis, only the infrared sensors and the camera are used in order to solve the maze problem. The dimensions of the robot and the configuration of the sensors are depicted in figure 4.1.

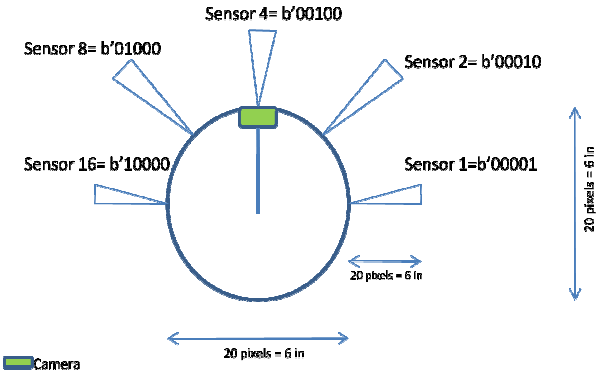

Figure 4.1. Robot Sensors Configuration

A typical configuration of the maze is illustrated in figure 4.2.

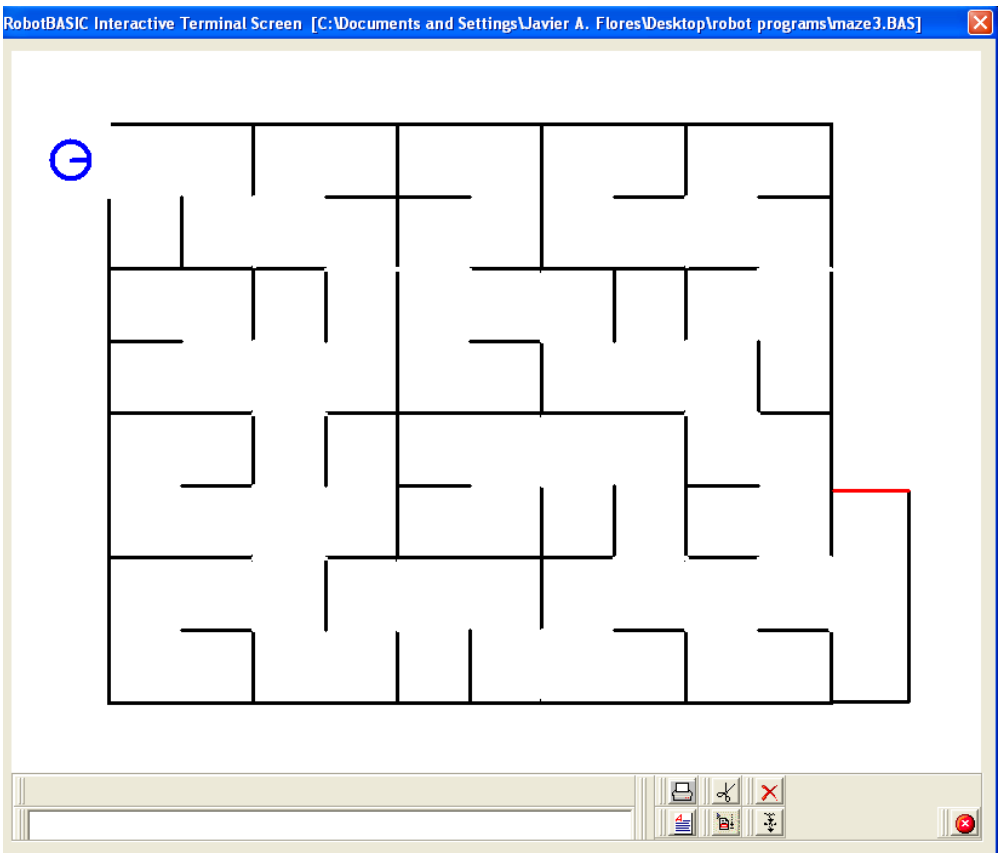

Figure 4.2. A 4×5 cell-maze.

The dimensions of the display terminal are 800 by 600 pixels. Taking these dimensions into account, the a maze of 640 by 480 pixels is generated, as illustrated in figure 4.2. The maze is composed of five cells by four cells giving a total of twenty cells. It is important to mention that there is only one path to the goal, which is identified by a red edge, which contrasts with all the other edges which are black. The infrared sensors have a range of approximately 20 pixels, so the maze is created accordingly: this determines placement of the walls. The camera is pointed in the direction the robot is heading; the intention of this camera is to provide a value corresponding to the color of an obstacle that it sees, instead of analyzing a picture. In this manner, the goal, being red, is easily identifiable.

4.2. CLASSICAL LOGIC SYSTEM IMPLEMENTATION

In order to get the robot to the goal an algorithm based on the fundamentals of classical logic was created. Each time the robot is in a new cell, the first thing it does is to count the number of exits that it senses and store the value in the variable named *cnt*, which is incremented each time it senses an obstacle. Table 4.1 illustrates the values of the possible exit directions of the current cell and will be store in a variable named *dir*.

Table 4.1 Look up table for possible exits in variable *dir*.

dir(decimal)	Binary Equivalent	Meaning
0	000	No exits
1	001	Exit only to right
2	010	Exit only fwd
3	011	Exit forward or right
4	100	Exit only to left
5	101	Exit only to left or right
6	110	Exit only to left or fwd
7	111	All three exits available

The variables *dir* and *cnt* are initialized to zero and starts to collect data of the possible exits in the current cell and the possible directions of them. There are a maximum of three possible exits: left, straight and right. Figure 4.3 illustrates how this is done.

```
numExits = 0
  dir = 0
  cnt = 0
  if not(rFeel()& 16)
    cnt = cnt+1
    dir = 4 // left side
  endif
  if not(rFeel() & 4)
    cnt = cnt+1
    dir = dir+2 // straight
  endif
  if not(rFeel() & 1)
    cnt = cnt+1
    dir = dir+1 // right side
  endif
```

Figure 4.3. Directions taken based on sensors data.

Then the robot performs in such a way that if encounters a cell with only one exit, the robot will take that exit. If it encounters a cell with multiple exists, then the robot will always try the left exit first, then forward, and lastly, the exit to its right. The robot will always go forward 60 pixels and then decide its next move accordingly. The latter can be seen in figure 4.4.

```
if cnt = 1
    if dir=4 then rTurn -90
    if dir=1 then  rTurn  90
    if dir=0 then rForward 60
    rForward 60
    continue
  endif
  if cnt = 0   // no exit, turn around and go back
    rTurn 180
    rForward 60
    continue
  endif
  //if there are multiple exits to this cell
  //(start with left, then CW)
    if dir & 4
      rTurn -90
      rForward 60
    else
      rForward 60 //if no exit to left, go fwd
    endif
```

Figure 4.4. If-statements to decide in zero and multiple exits

The principal idea of this algorithm is that it tries every possible option available to it when it encounters a cell with multiple exits. If all of the above logic is executed repeatedly, eventually the robot will get out of the maze, as this produces a method of exhaustive search. In the maze there is a cell with a red line, symbolizing the goal. The camera will always be looking for this object and it will not stop until it is found, leading to success: solution of the problem, or a path through the maze. The camera's search is executed with the following while loop:

```
while rLook() <> Red
        //statements are in here
wend
```

As a result of this approach, the robot performs thirty turns to complete the selected maze. Figures 4.5 through 4.28 illustrate the series of turns that leads to the goal denoted by a red line.

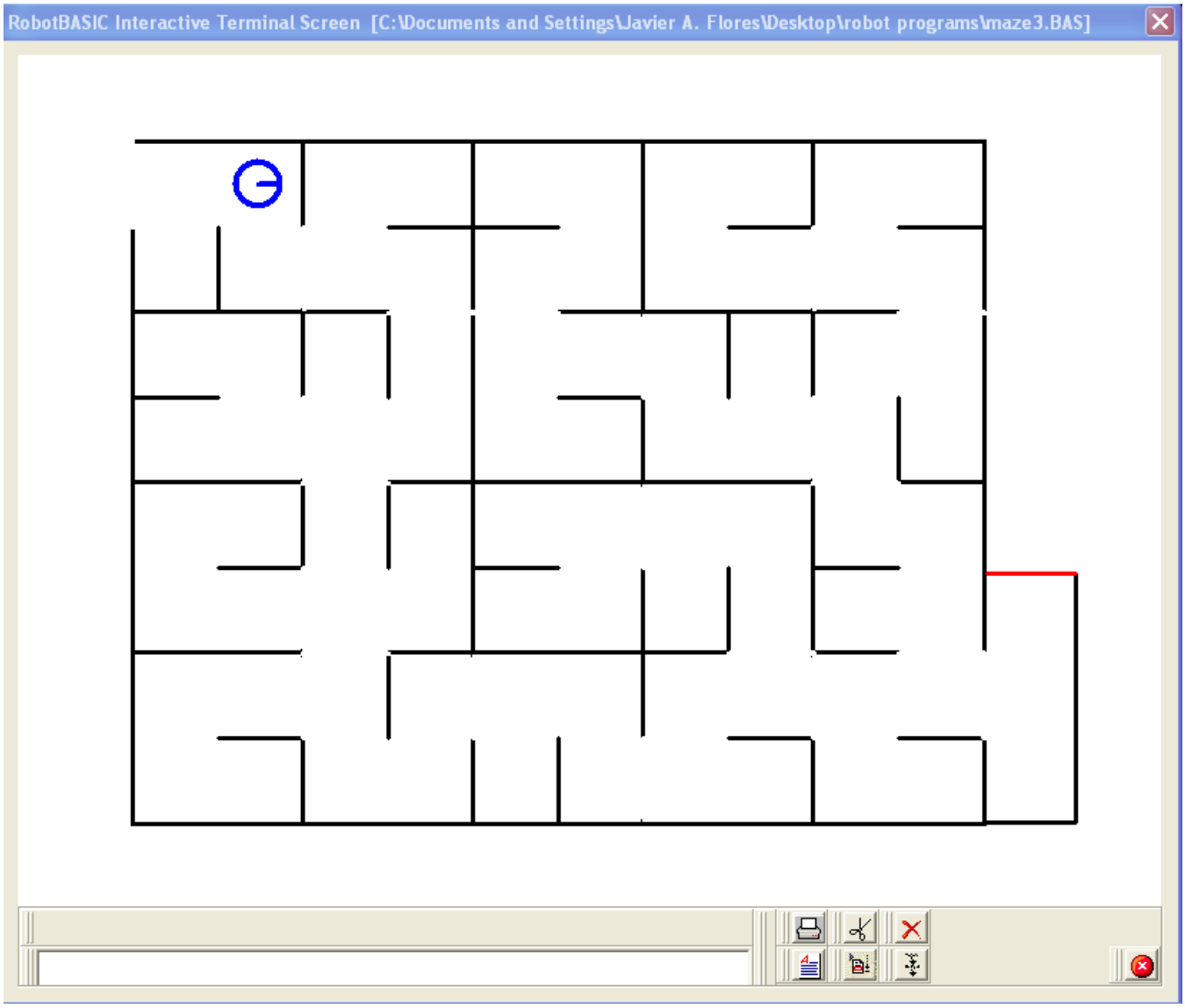

Figure 4.5. Executing classical logic algorithm

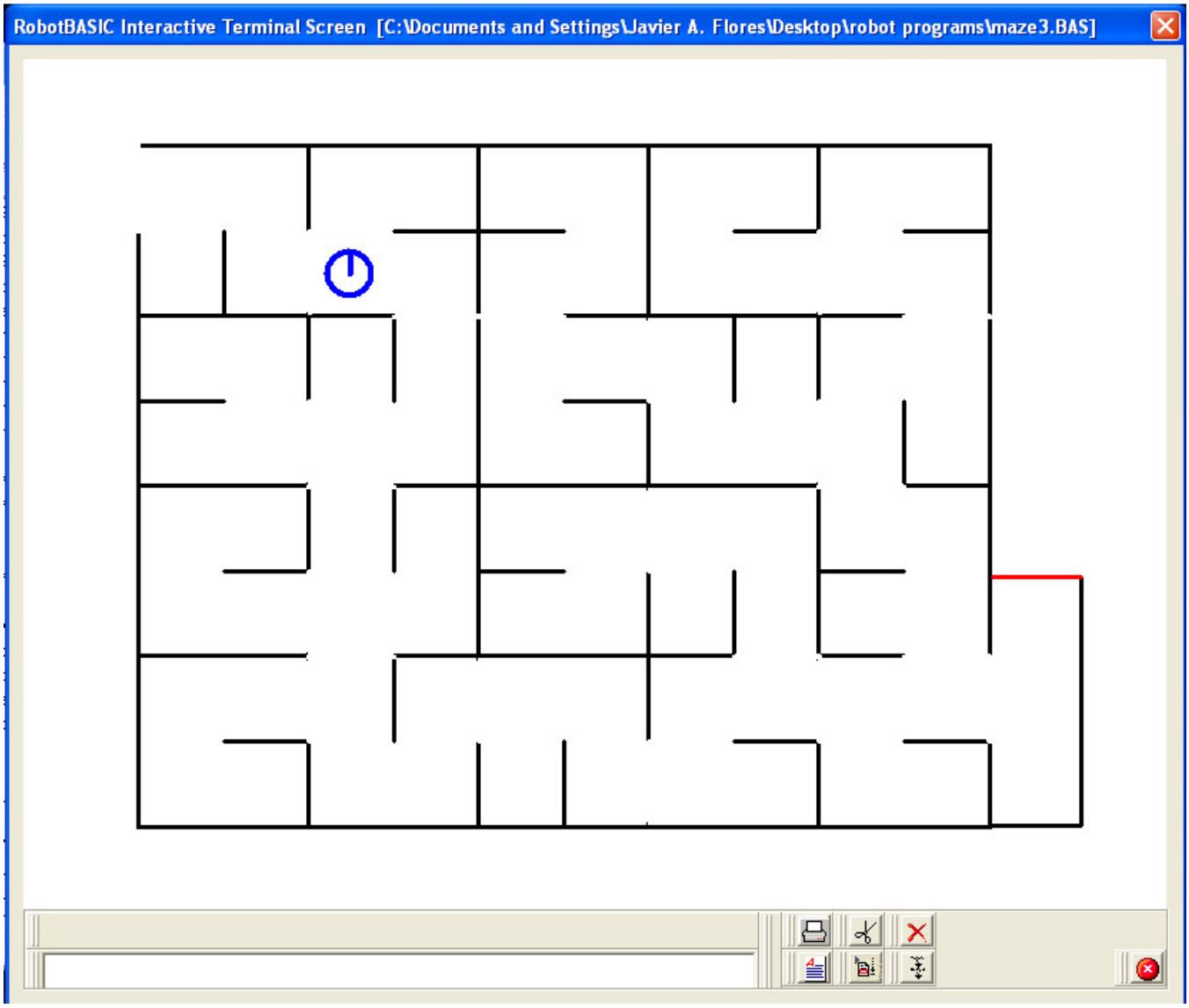

Figure 4.6. Executing classical logic algorithm

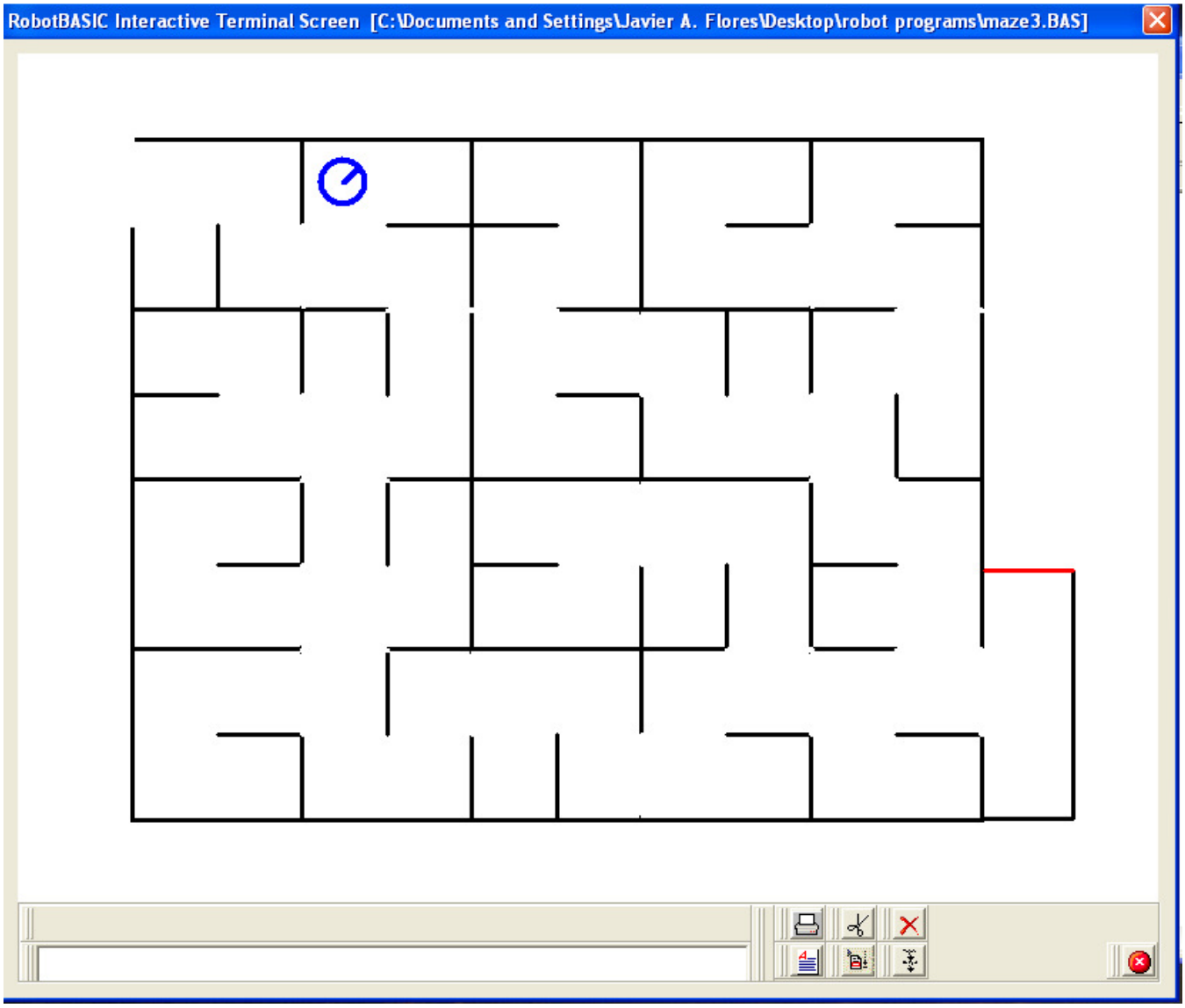

Figure 4.7. Executing classical logic algorithm

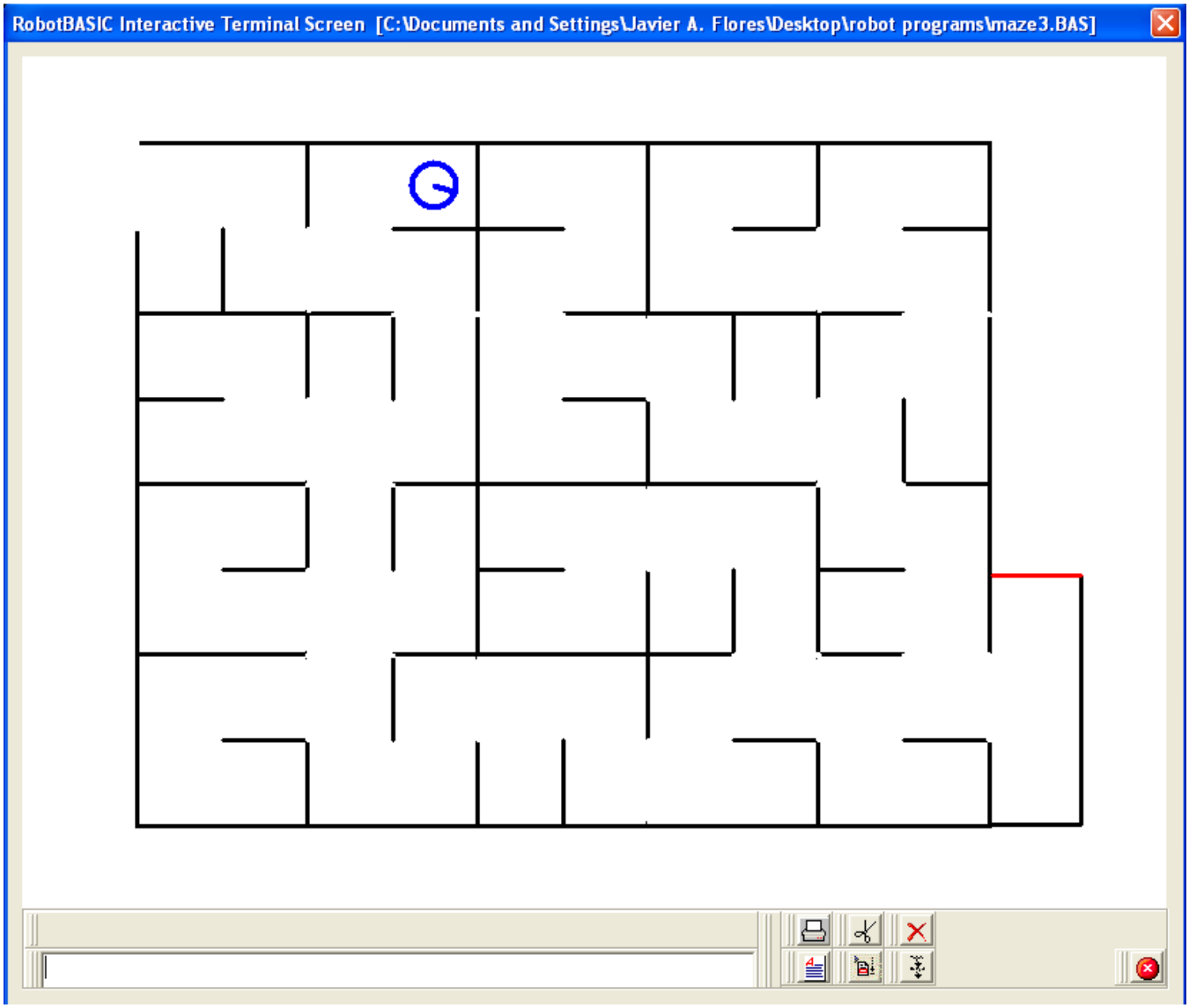

Figure 4.8. Executing classical logic algorithm

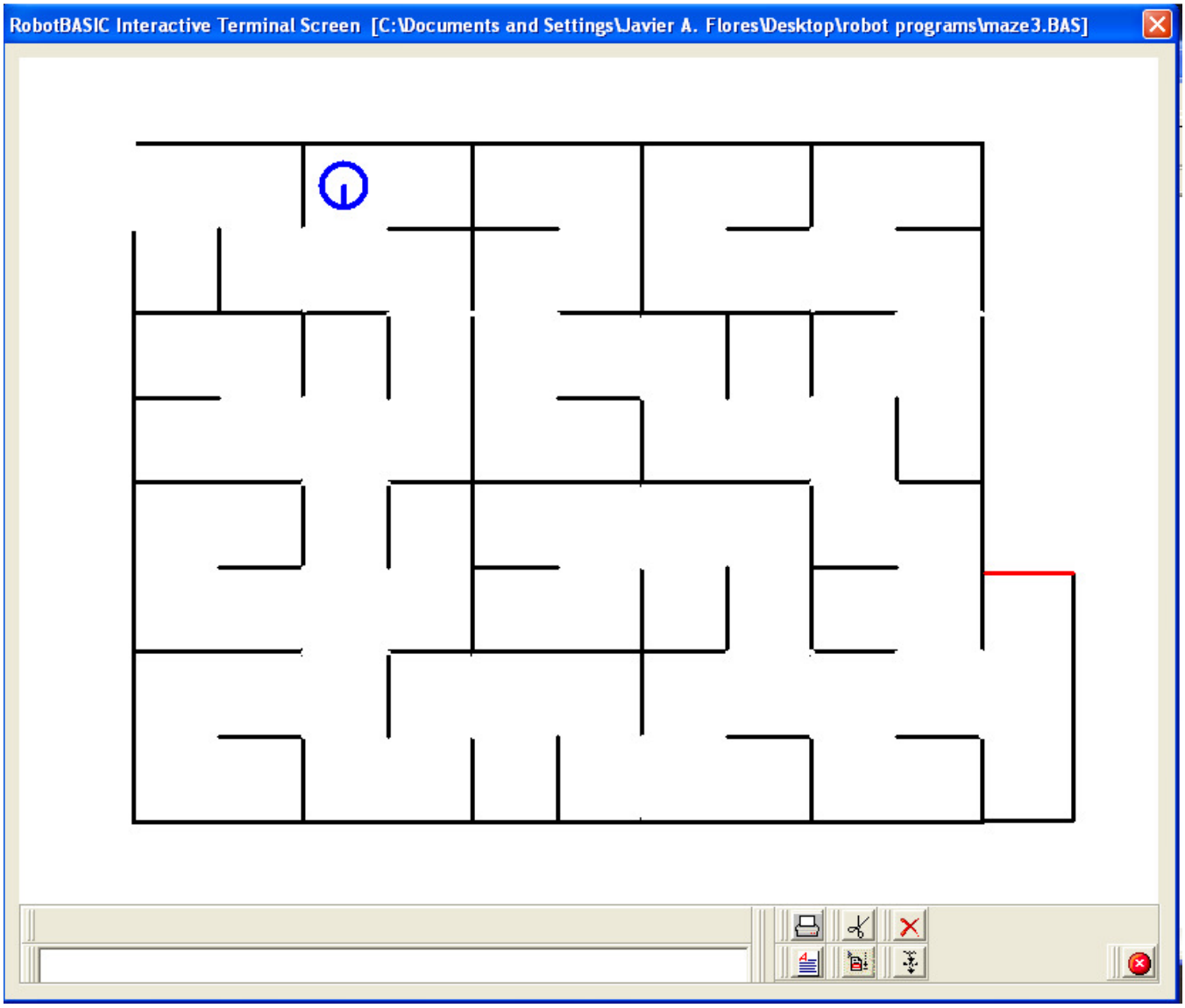

Figure 4.9. Executing classical logic algorithm

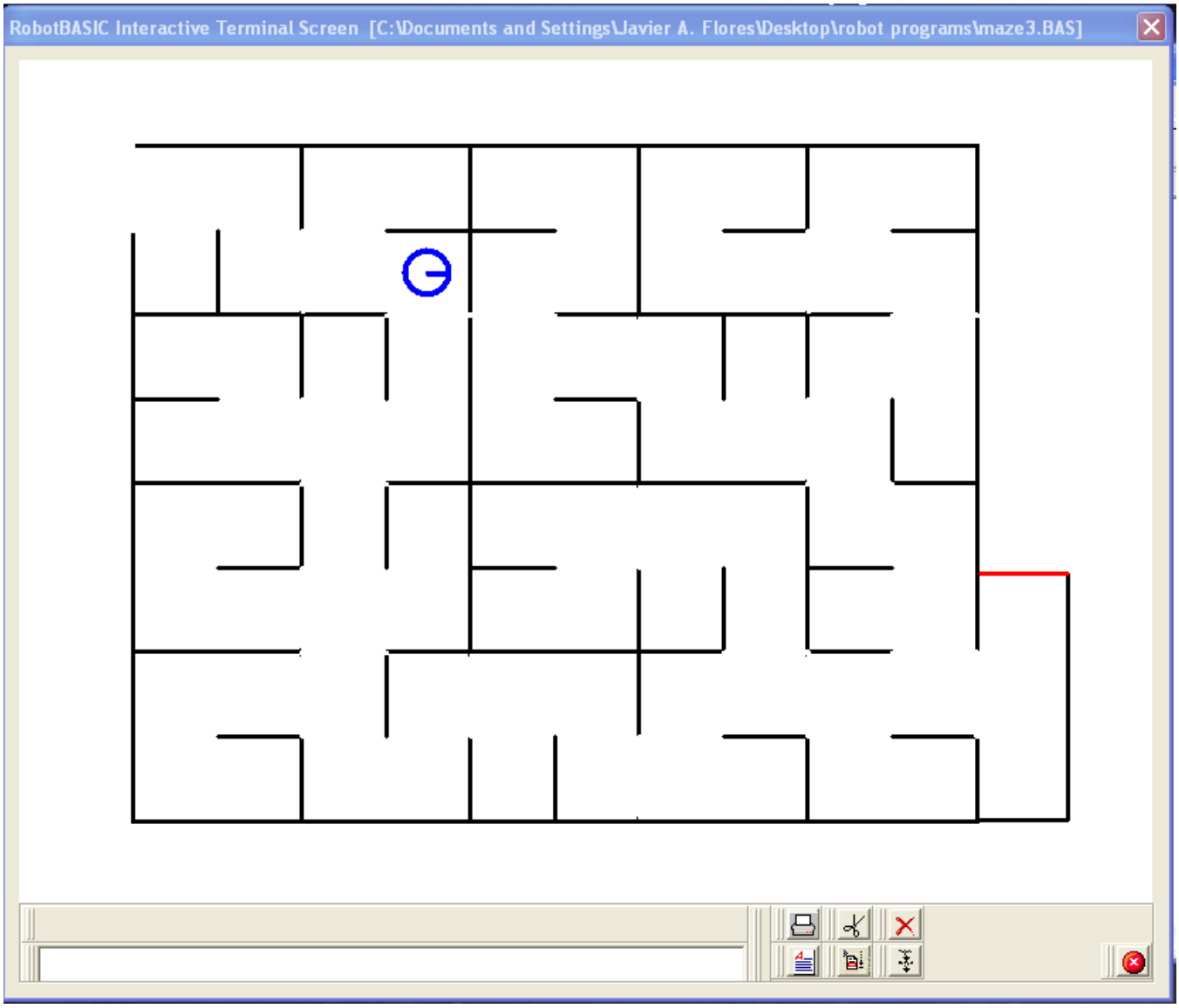

Figure 4.10. Executing classical logic algorithm

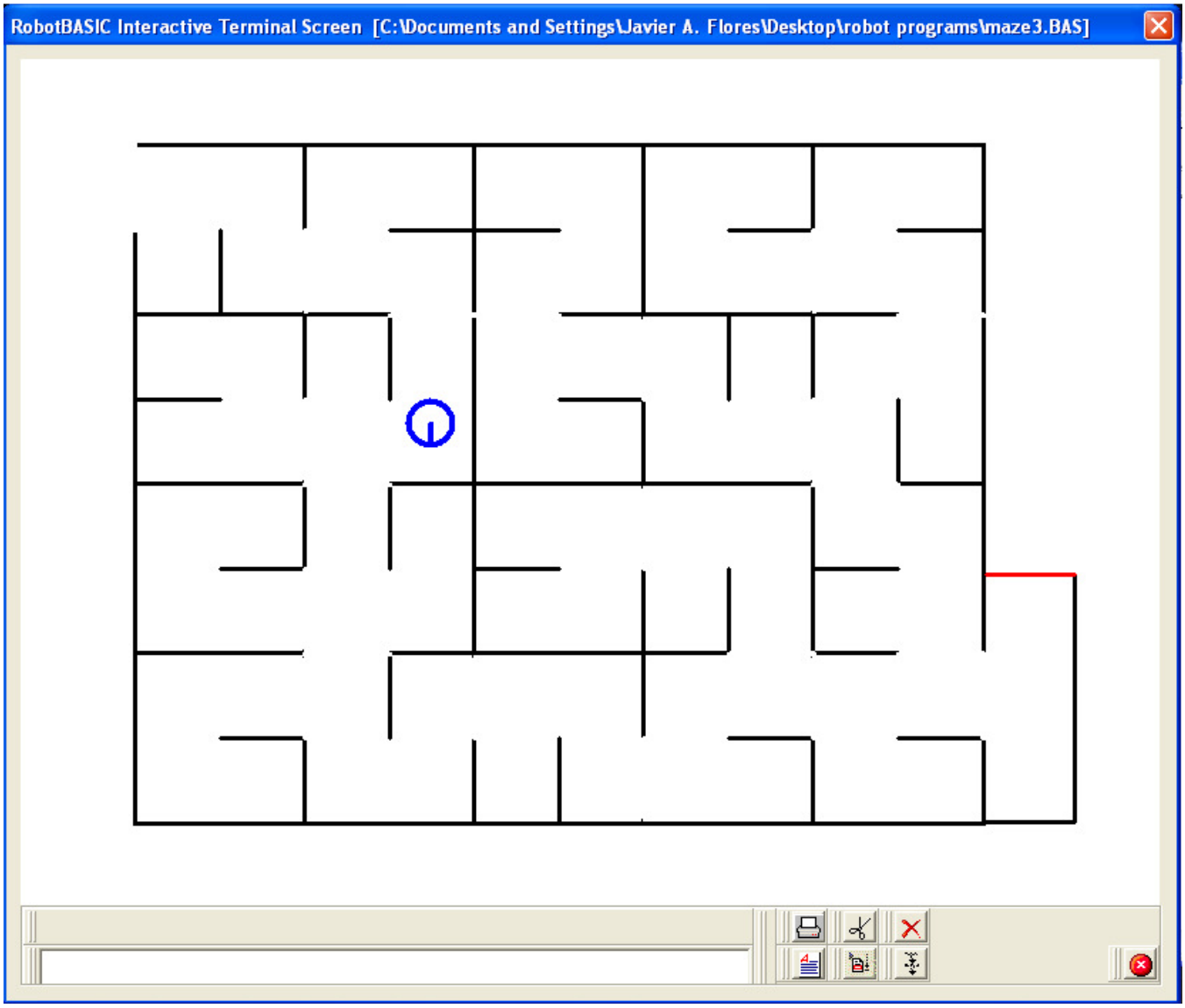

Figure 4.11. Executing classical logic algorithm

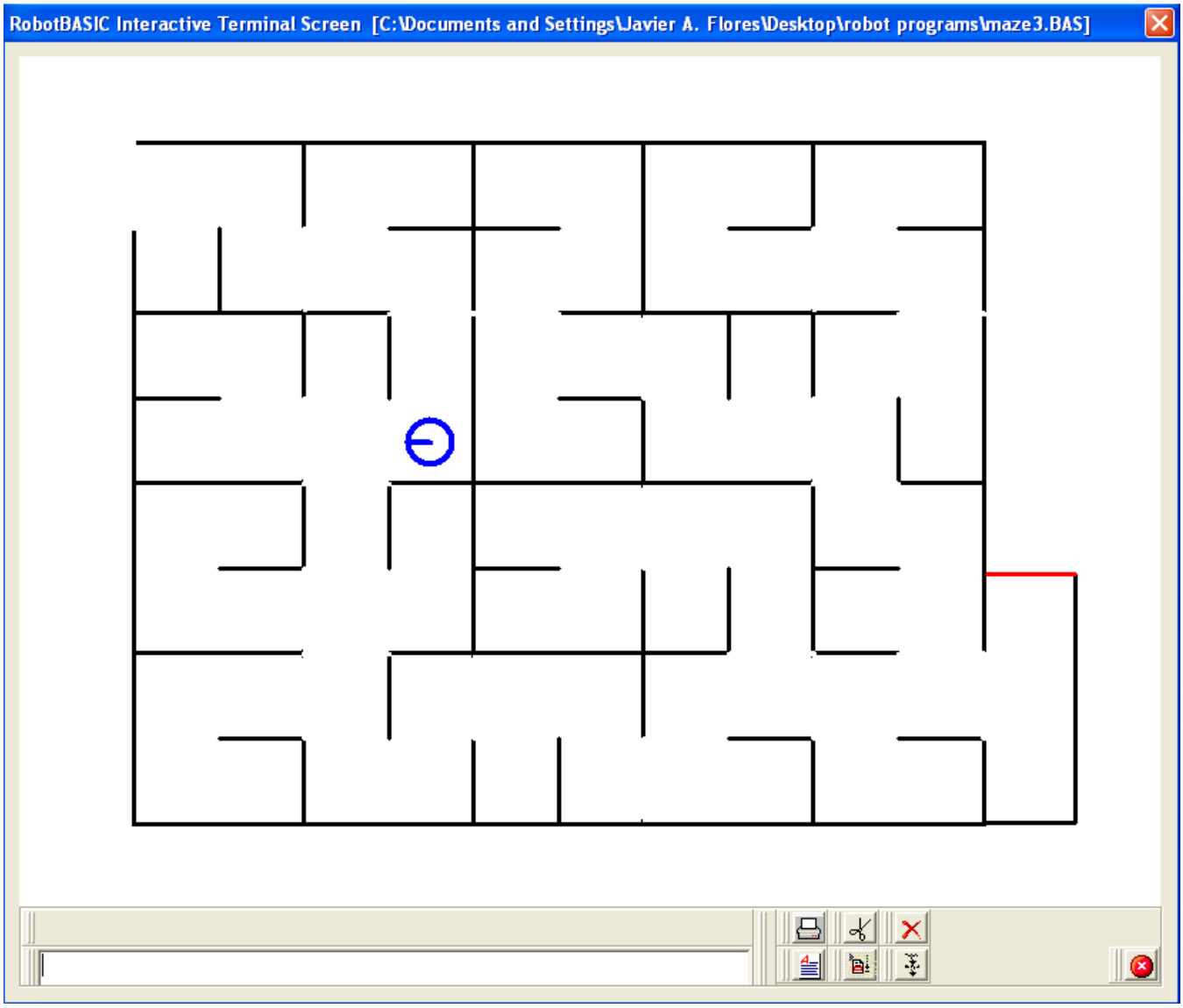

Figure 4.12. Executing classical logic algorithm

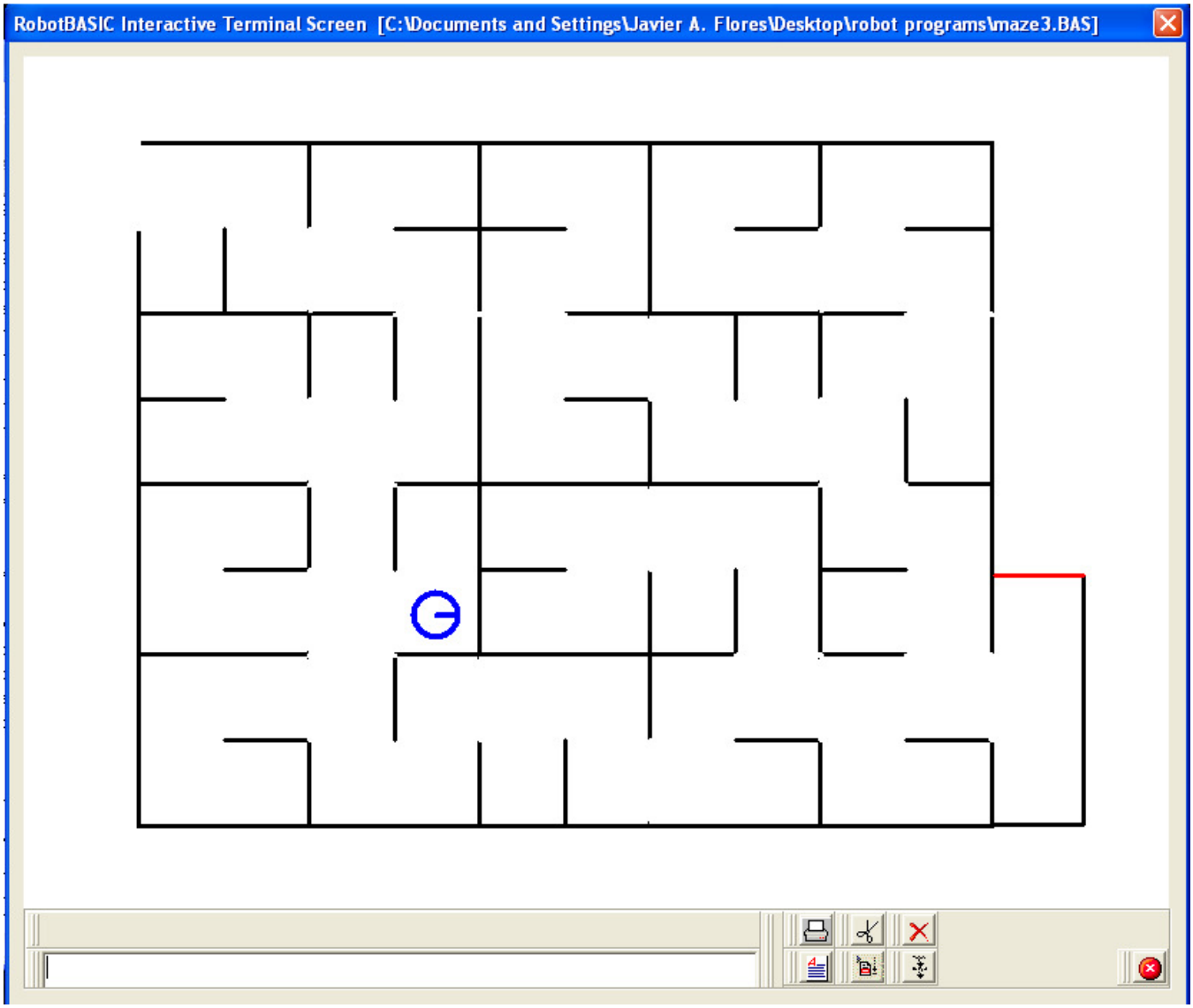

Figure 4.13. Executing classical logic algorithm

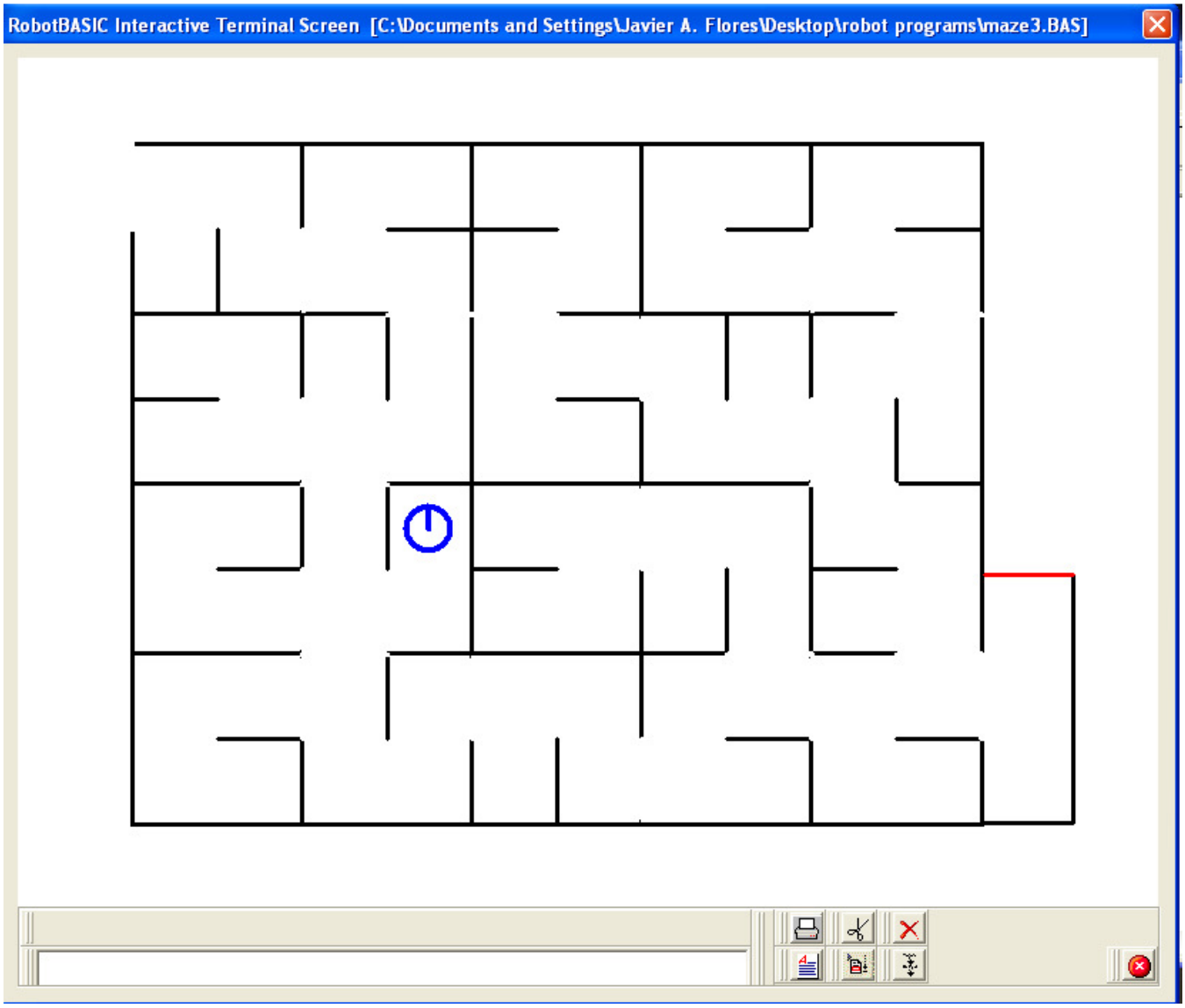

Figure 4.14. Executing classical logic algorithm

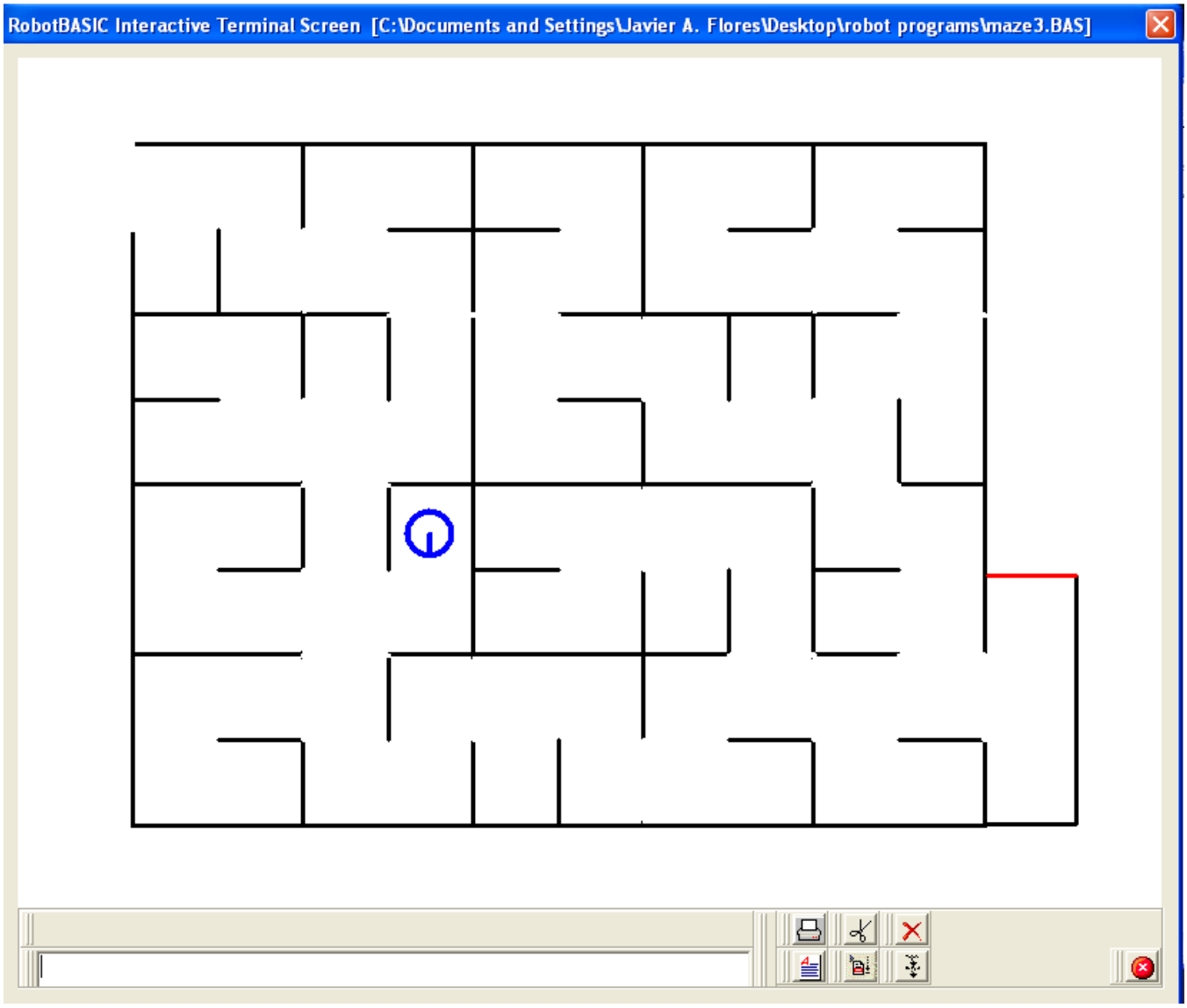

Figure 4.15. Executing classical logic algorithm

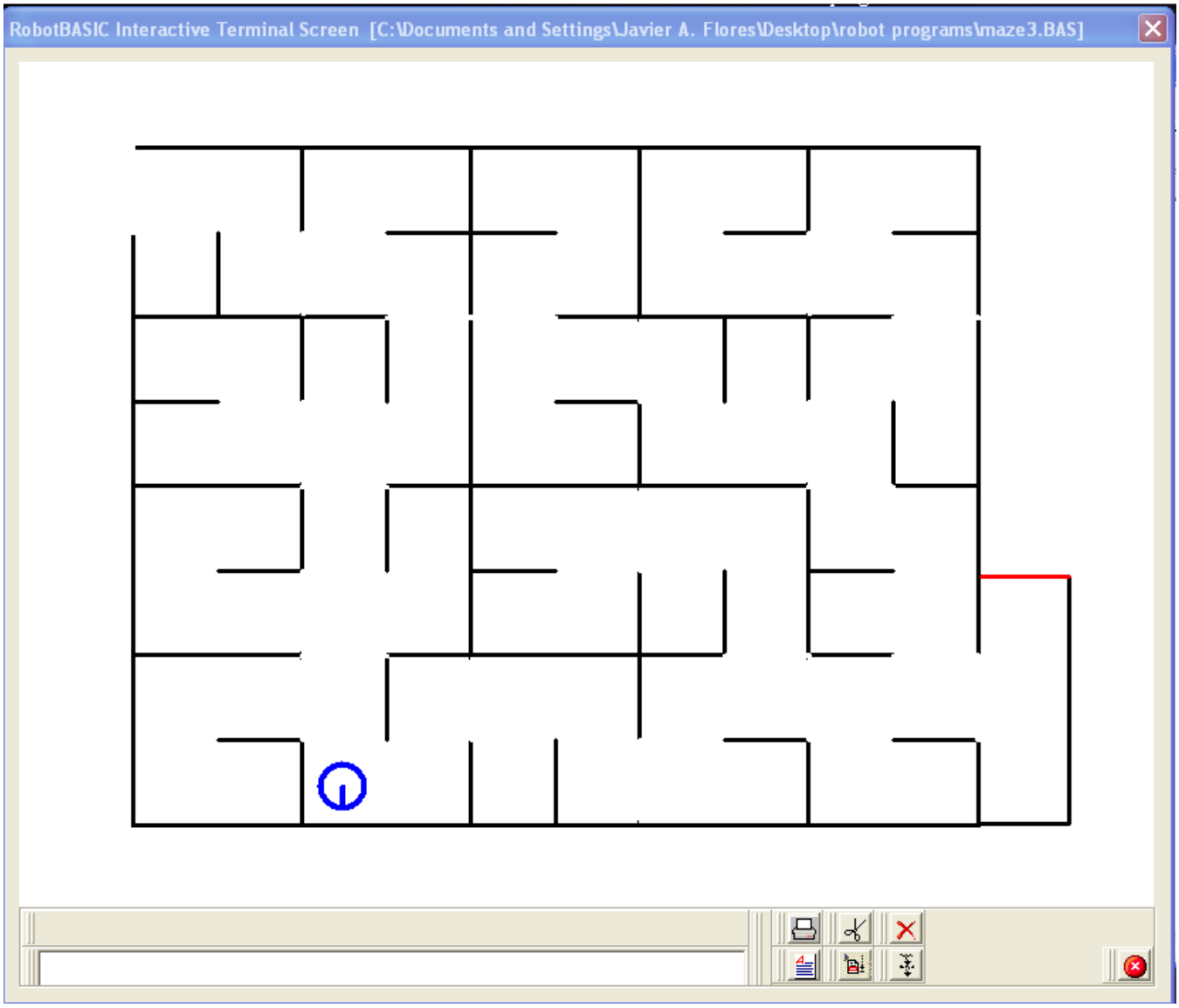

Figure 4.16. Executing classical logic algorithm

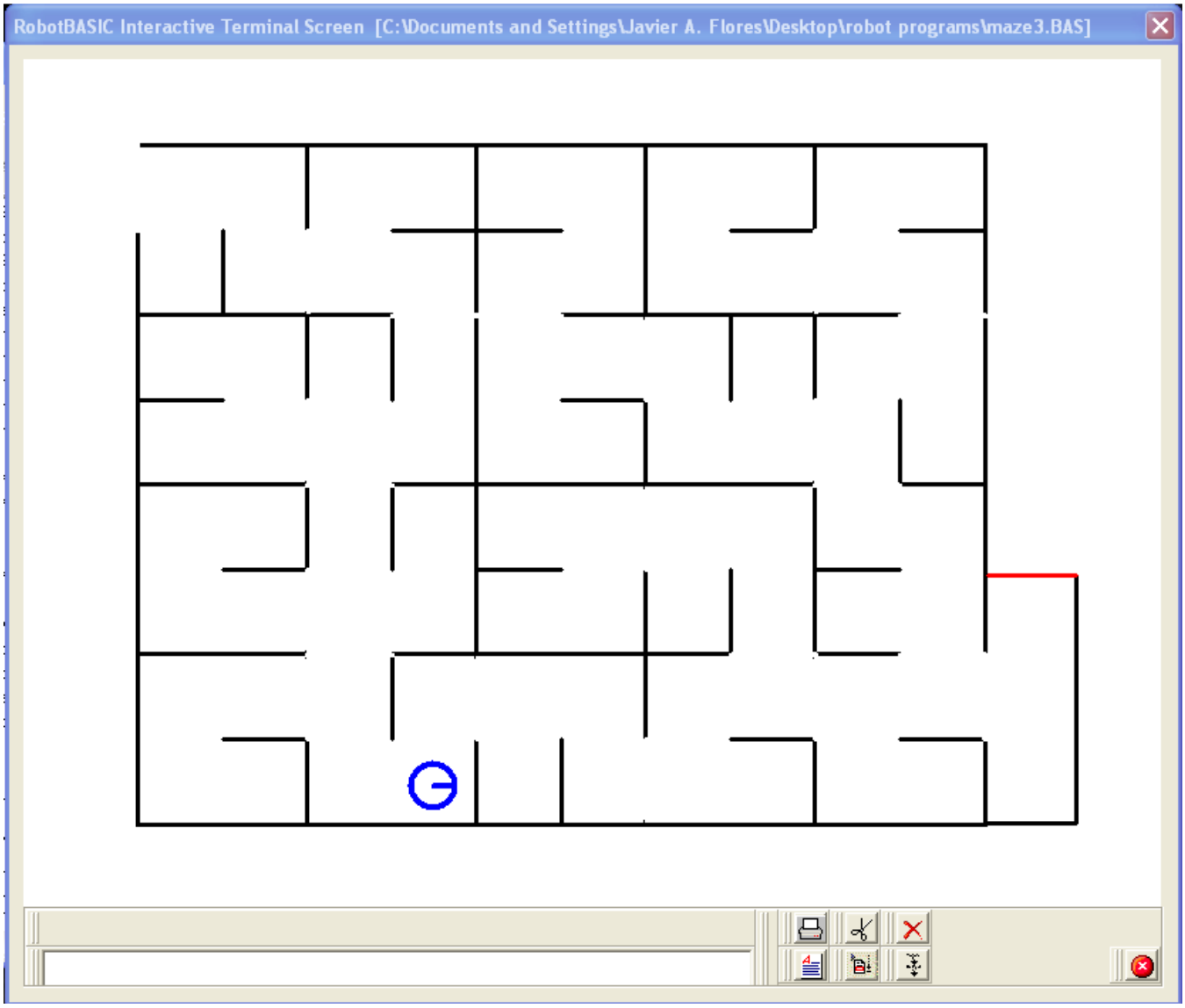

Figure 4.17. Executing classical logic algorithm

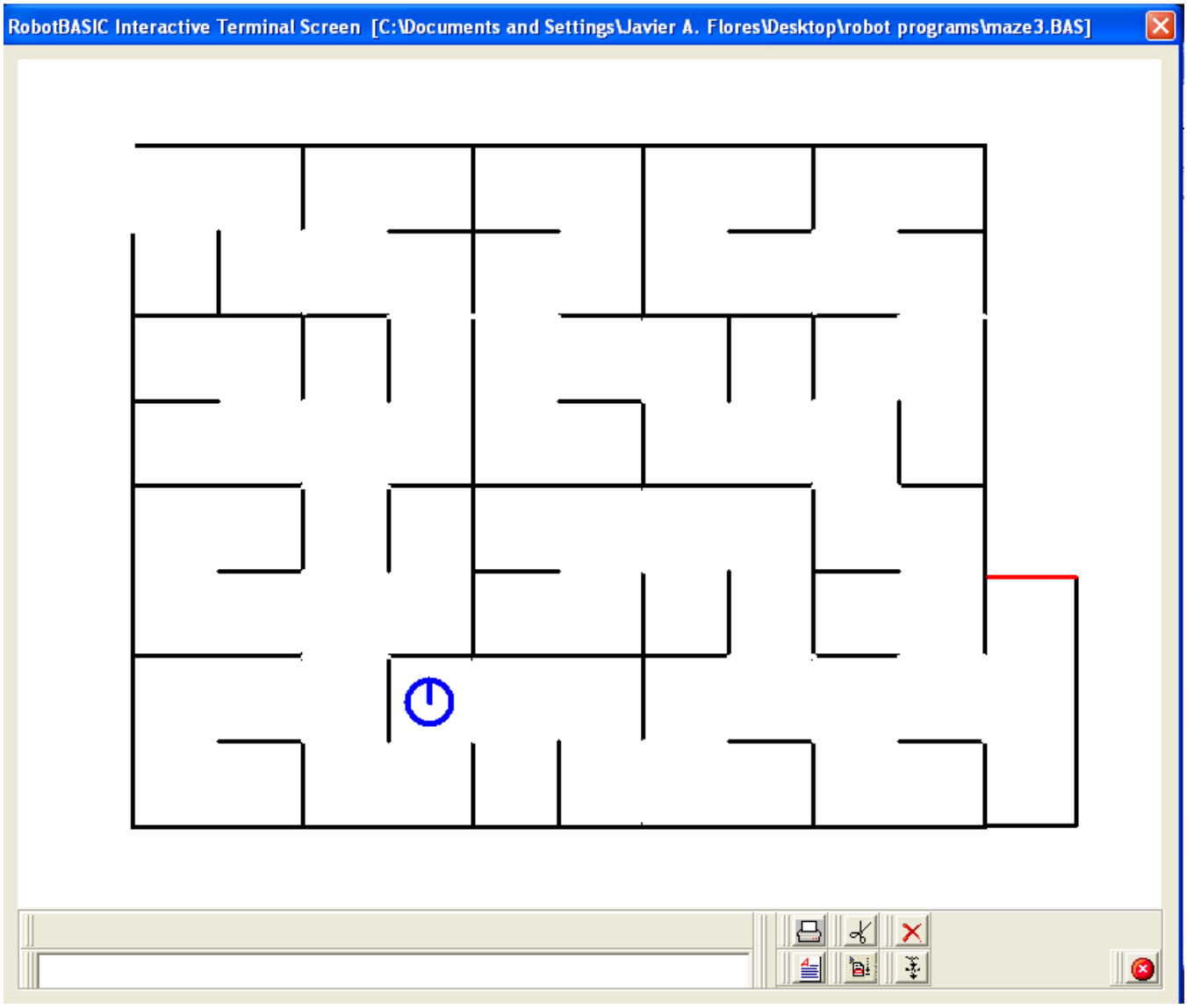

Figure 4.18. Executing classical logic algorithm

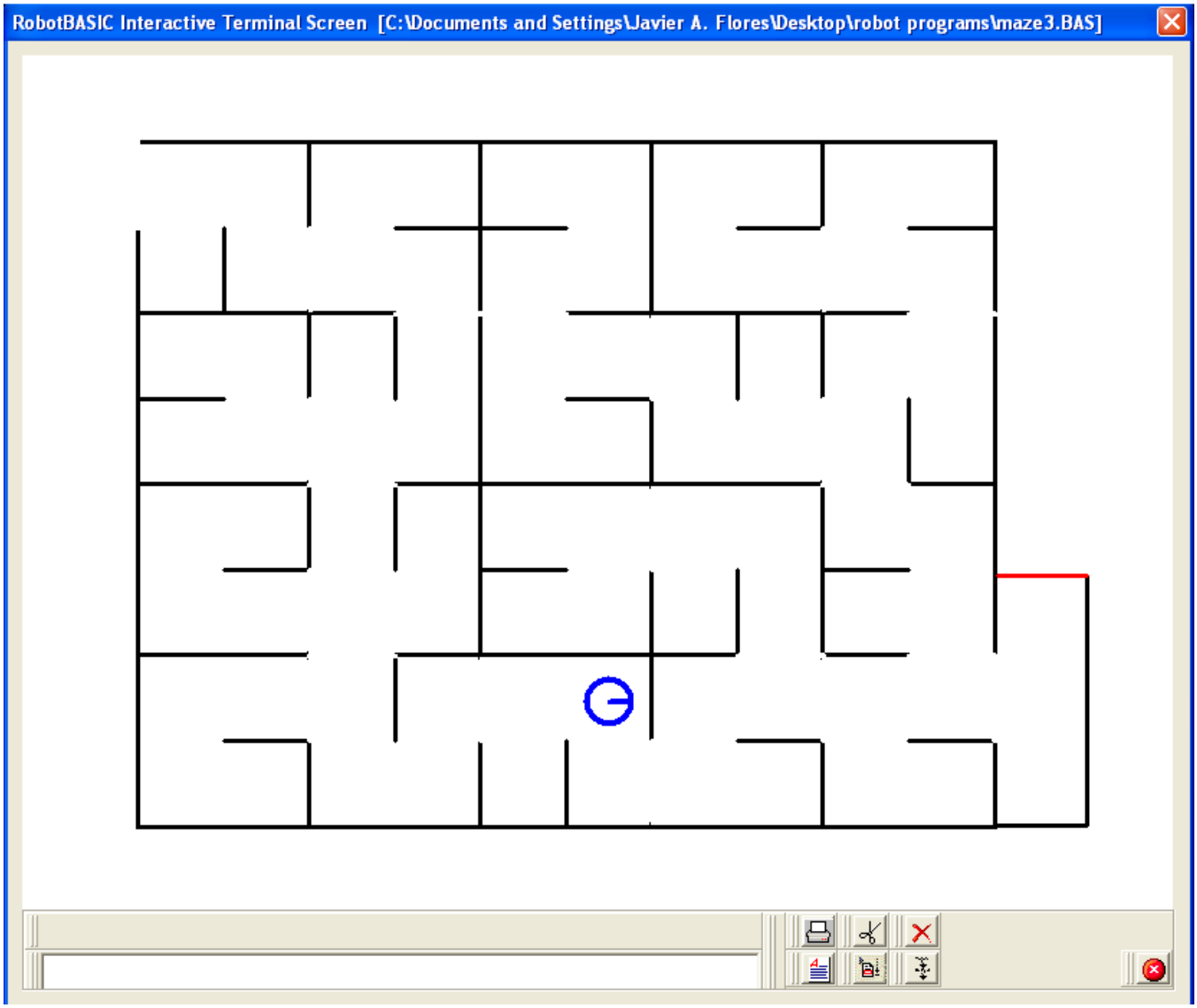

Figure 4.19. Executing classical logic algorithm

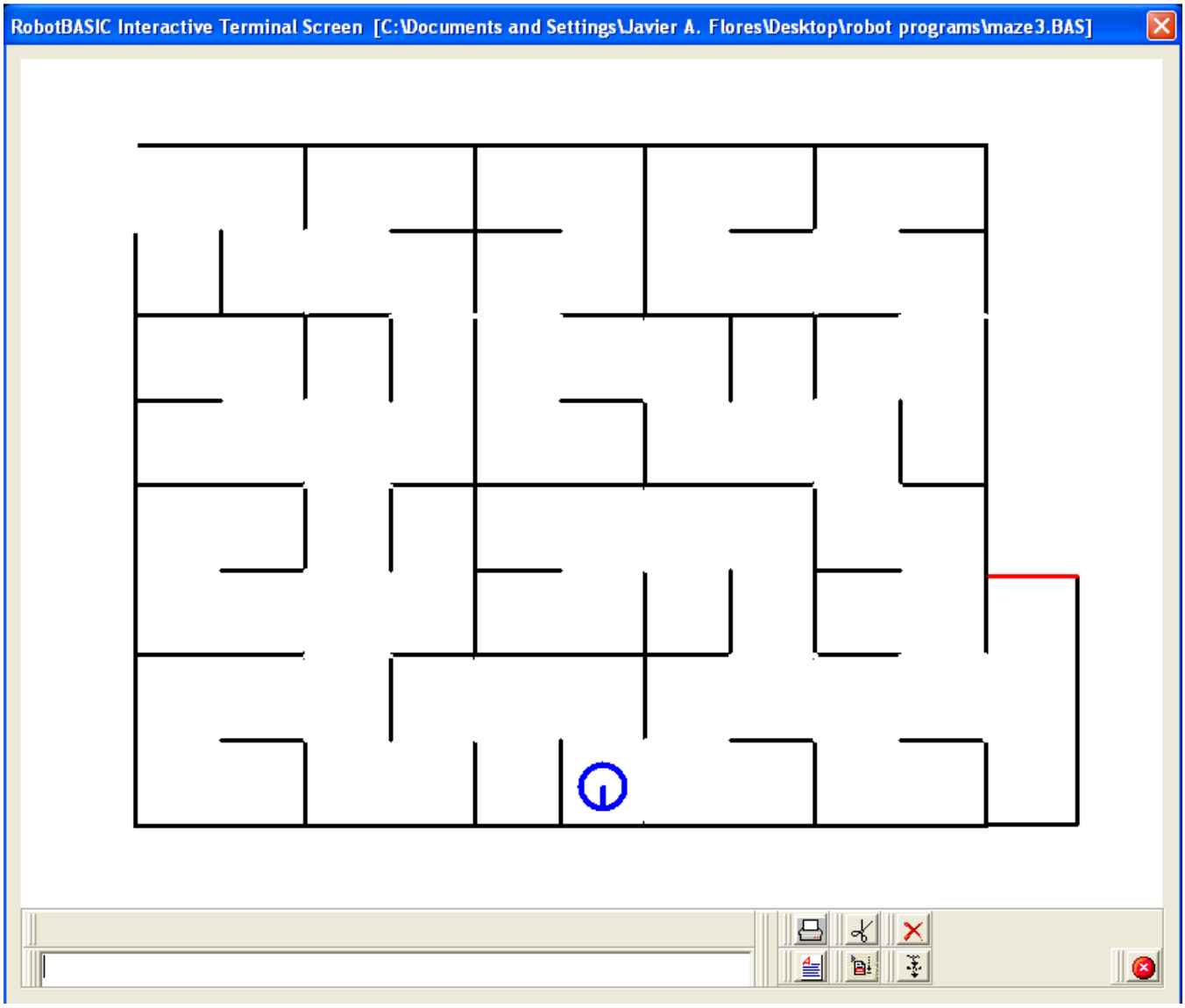

Figure 4.20. Executing classical logic algorithm

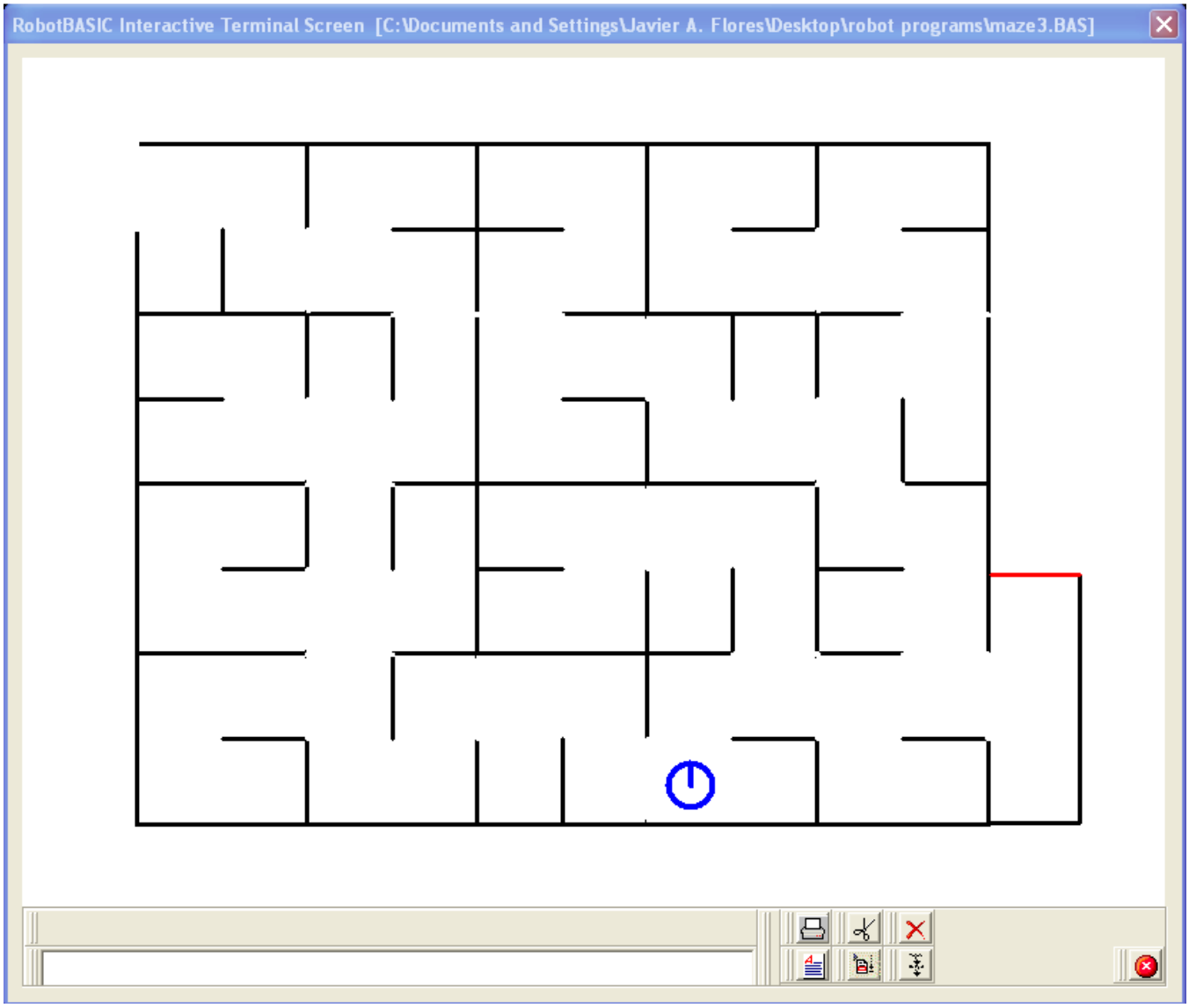

Figure 4.21. Executing classical logic algorithm

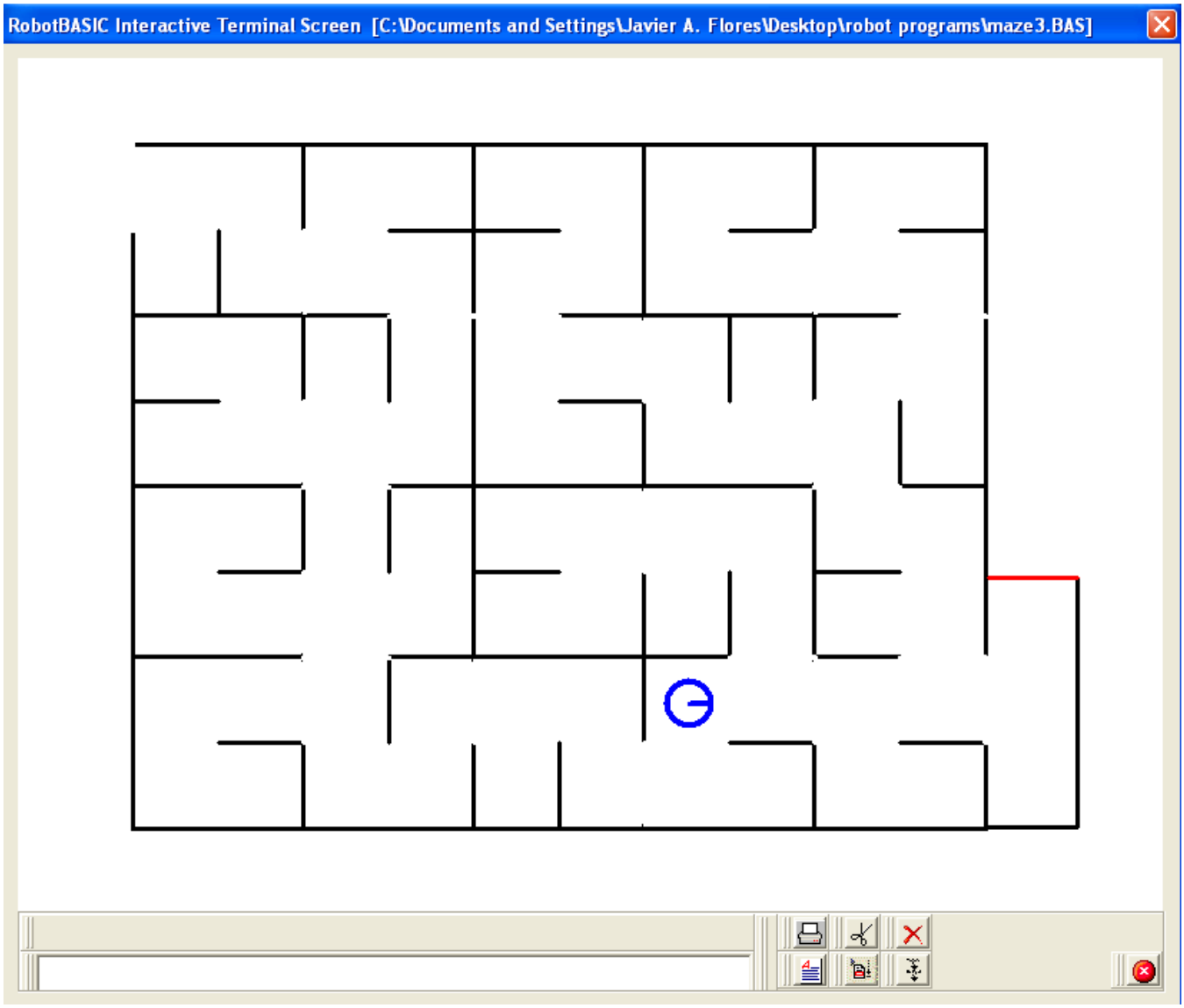

Figure 4.22. Executing classical logic algorithm

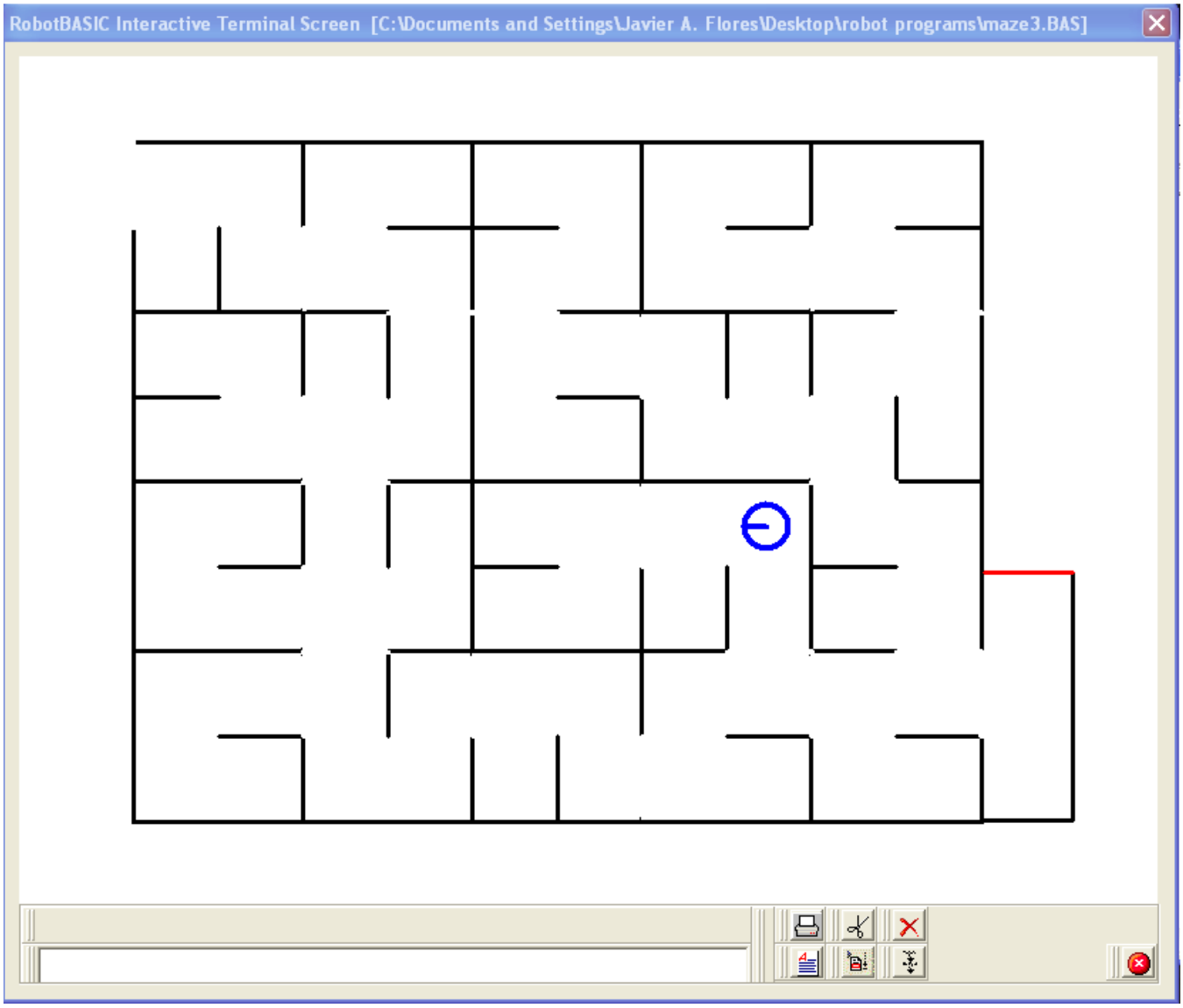

Figure 4.23. Executing classical logic algorithm

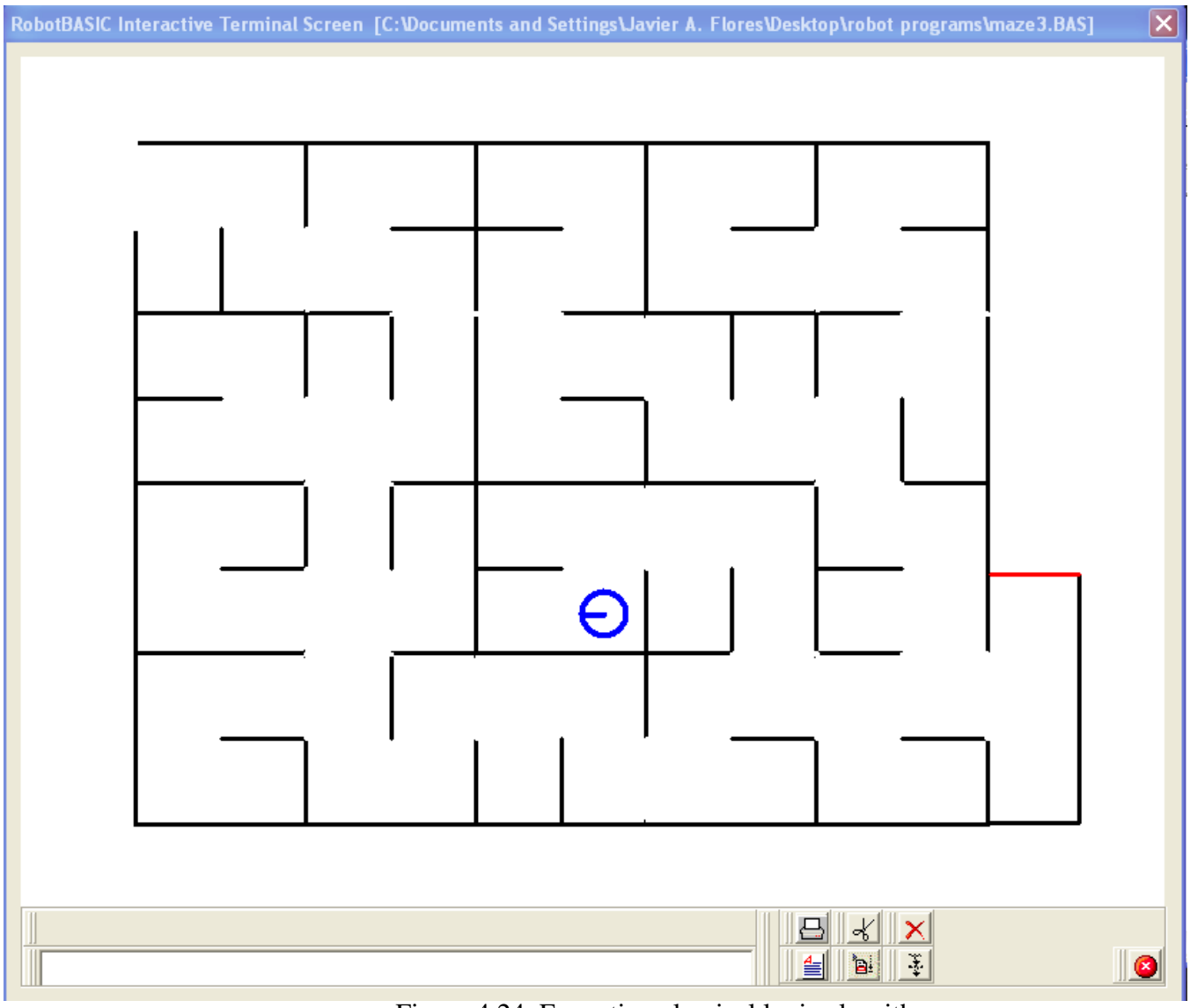

Figure 4.24. Executing classical logic algorithm

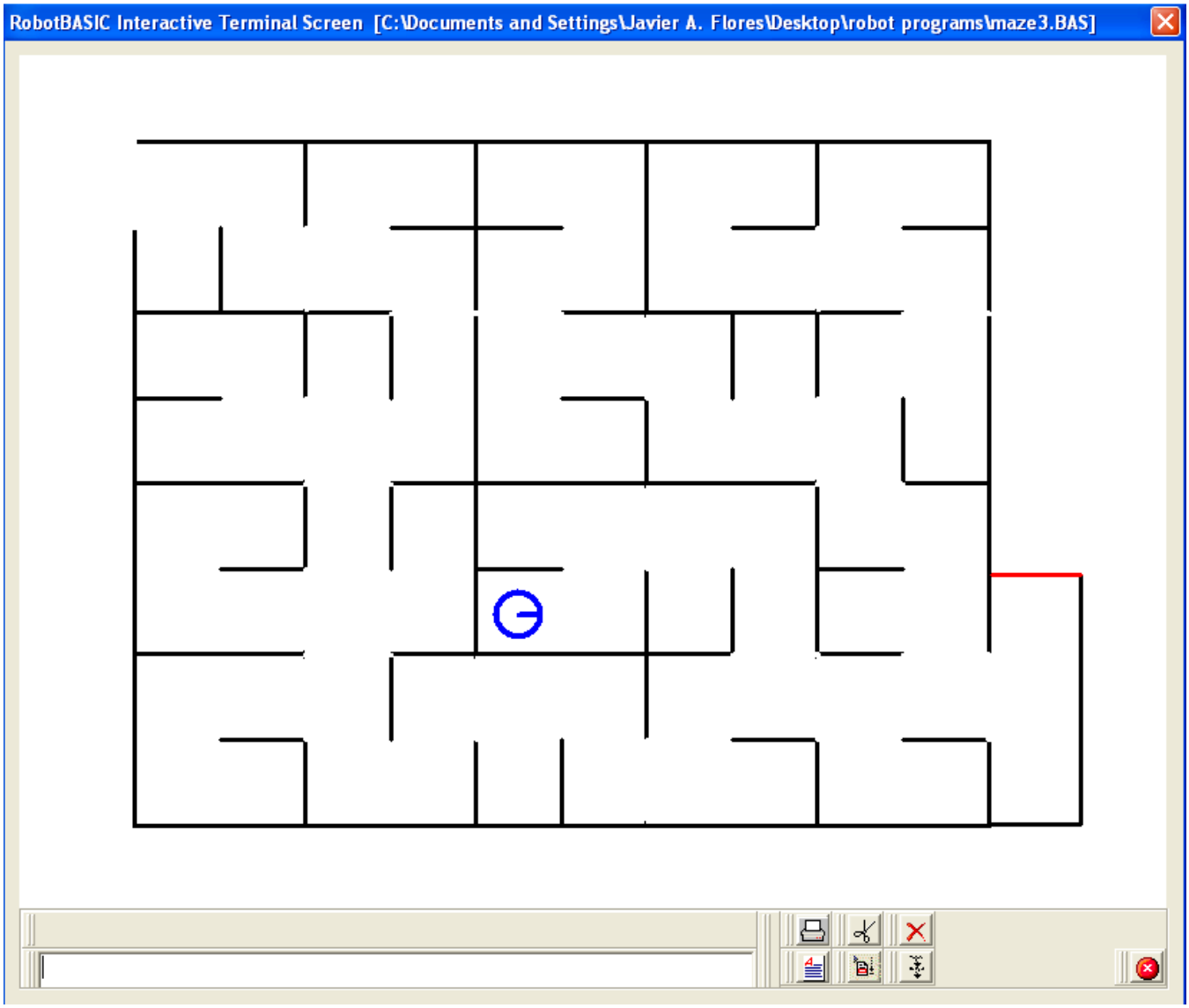

Figure 4.25. Executing classical logic algorithm

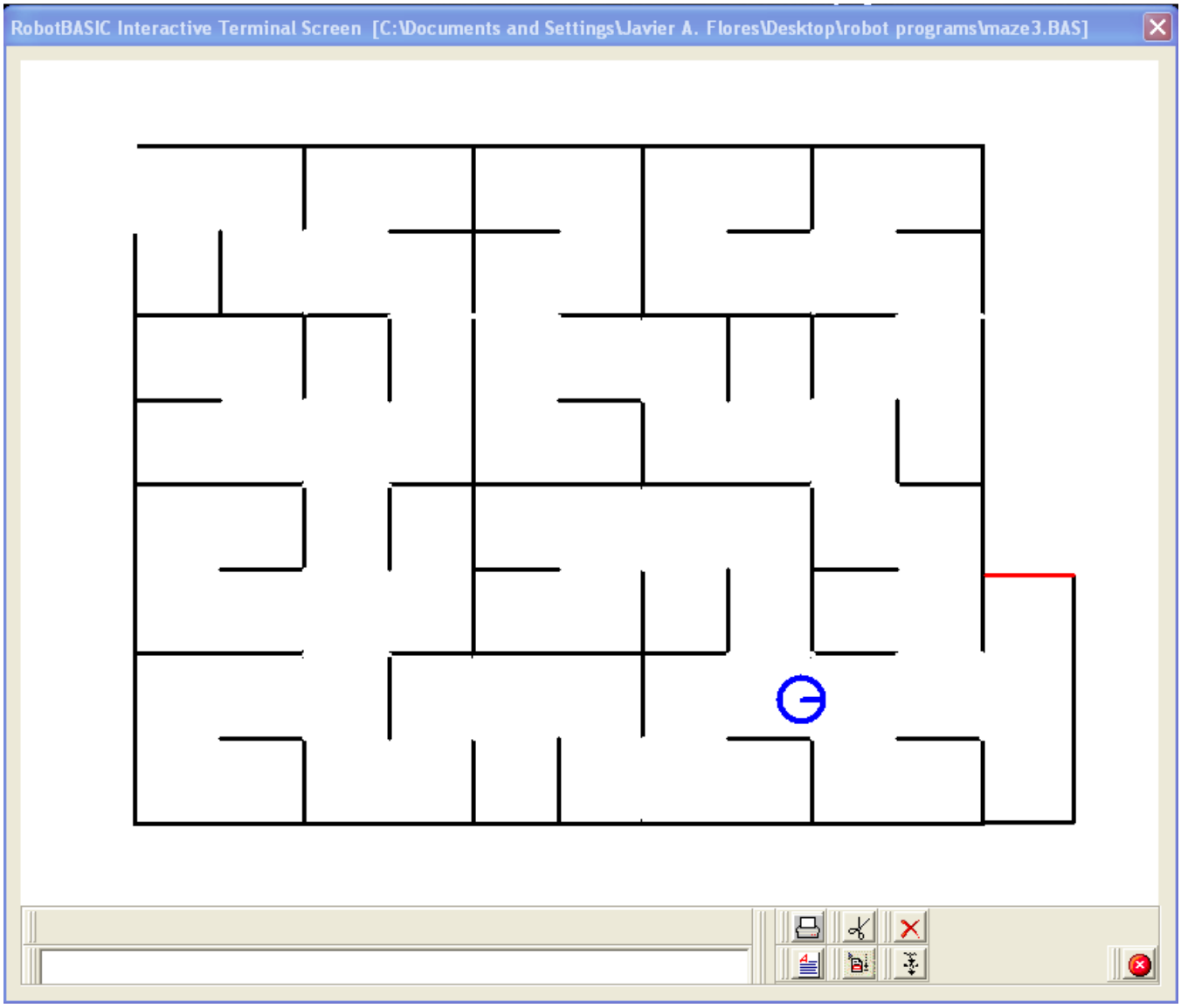

Figure 4.26. Executing classical logic algorithm

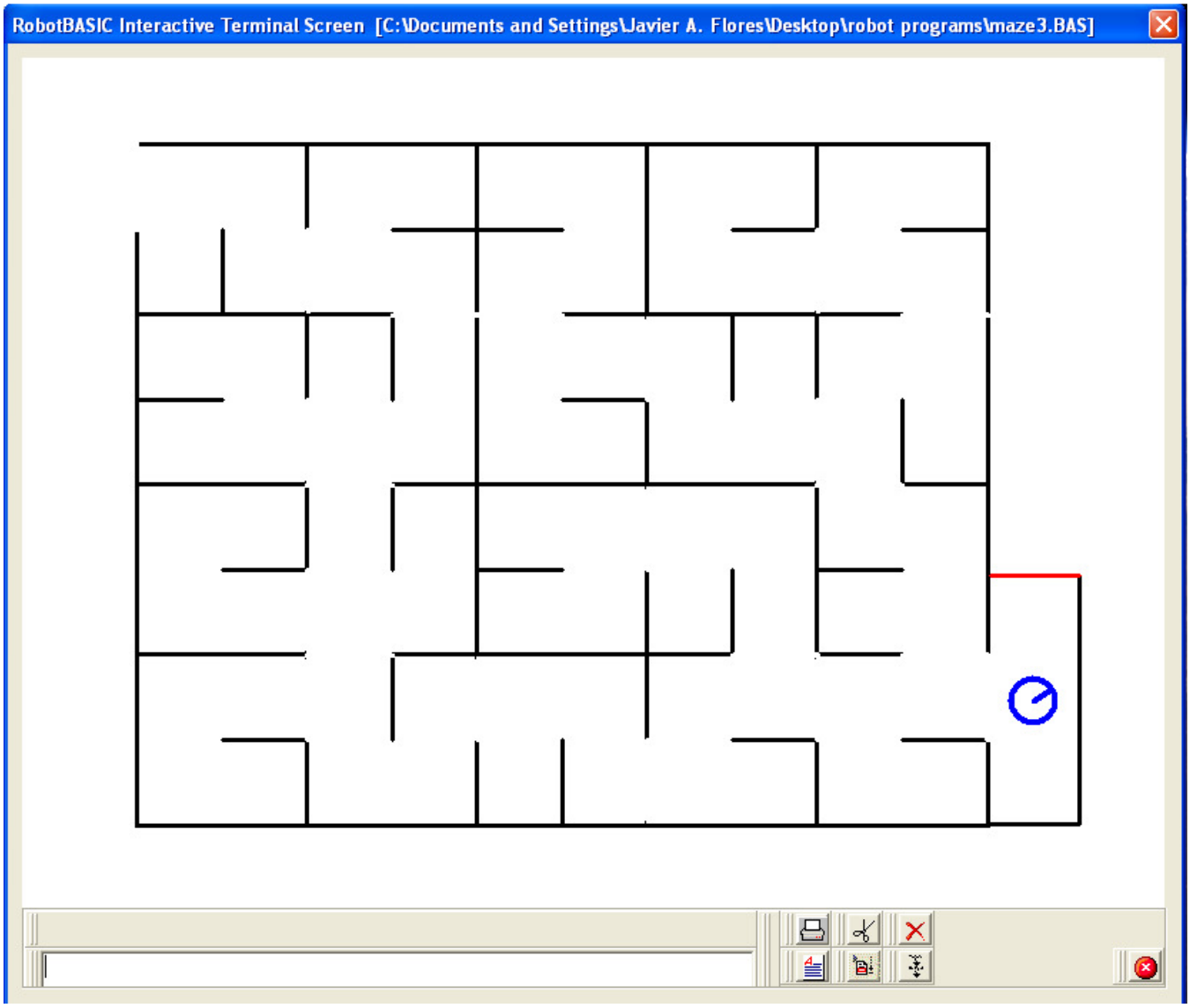

Figure 4.27. Executing classical logic algorithm

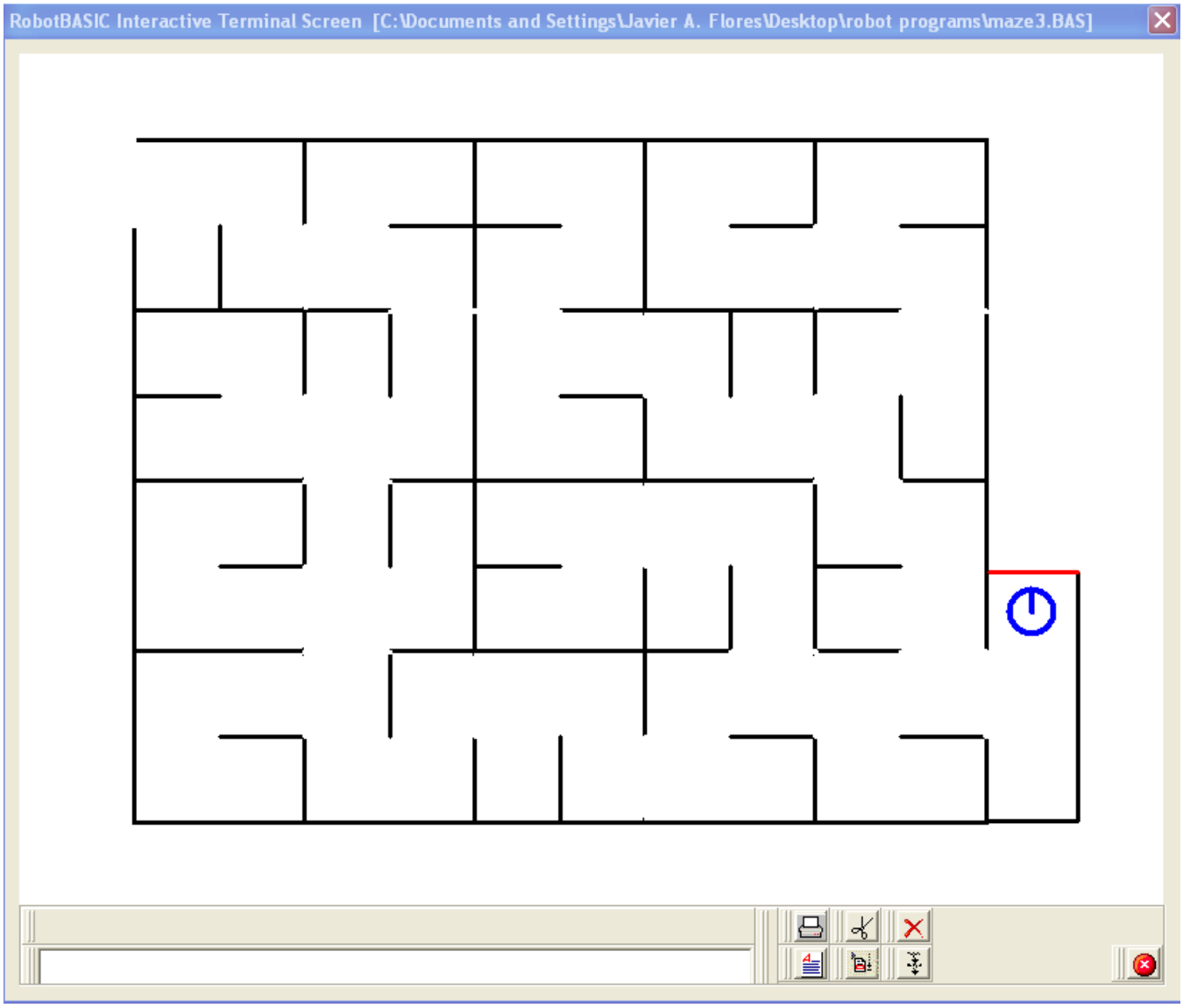

Figure 4.28. Executing classical logic algorithm

4.3 ARTIFICIAL NEURAL NETWORK IMPLEMENTATION

In the previous section, it was demonstrated that by executing a classical logic algorithm, a robot can negotiate obstacles, as well as correcting wrong turns, to reach a goal with the aid of a camera and five infrared sensors. This section will contrast this result with results from a robot, with the same sensors previously used, that employs an artificial neural network algorithm to navigate the same maze.

As explained in chapter two, a neural network topology must be chosen for the given problem. Through literature review, it has been shown that a back-propagation training with supervised learning leads to a very accurate approximation when dealing with uncertainty. In order to optimize a neural network with such learning, the net must be provided with a target vector. A target vector, in this particular case, will be the different reactions to the different possible scenarios when encountering an obstacle. Figure 4.29 illustrates a set of seven different possible scenarios within the maze.

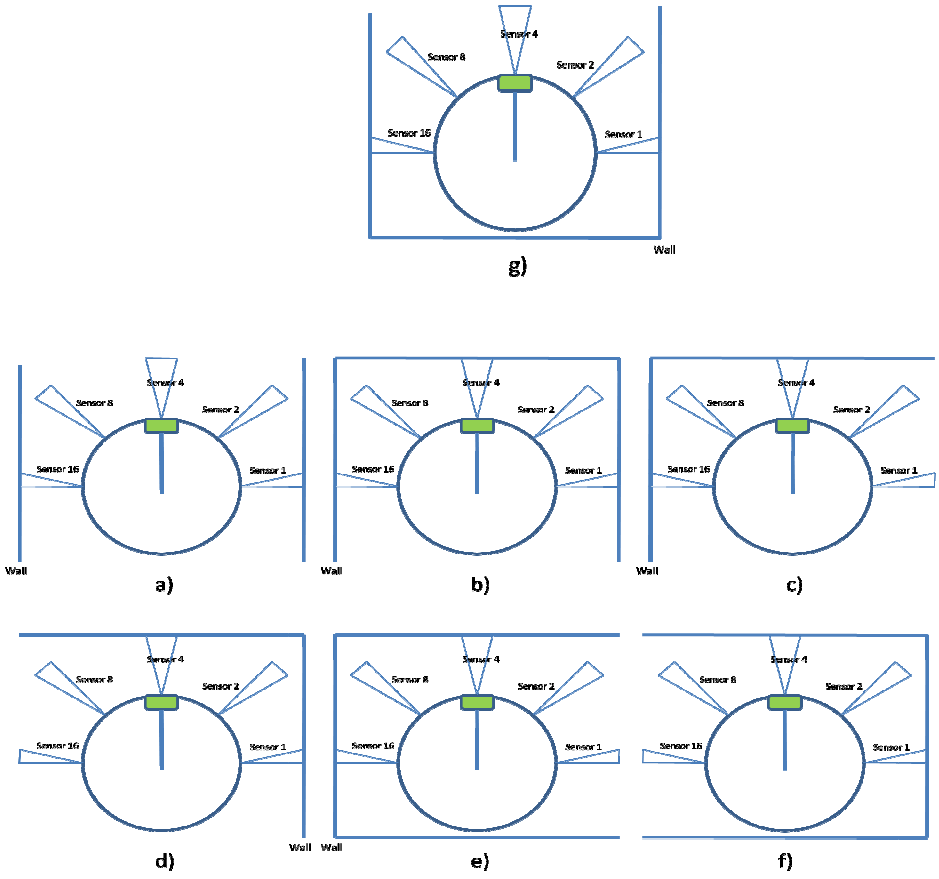

Figure 4.29. Different possible scenarios when negotiating a maze

Once the seven possible scenarios are encoded, a neural network can be implemented. Figure 4.30 illustrates a three layer network having as follows: one input layer with seven neurons, one hidden layer with eight neurons and an output layer with five neurons. The net is constructed such that it is being fed with five infrared sensors plus a camera, and has five outputs, corresponding to "left," "right," "turn 180^{o}," "forward" and "goal reached."

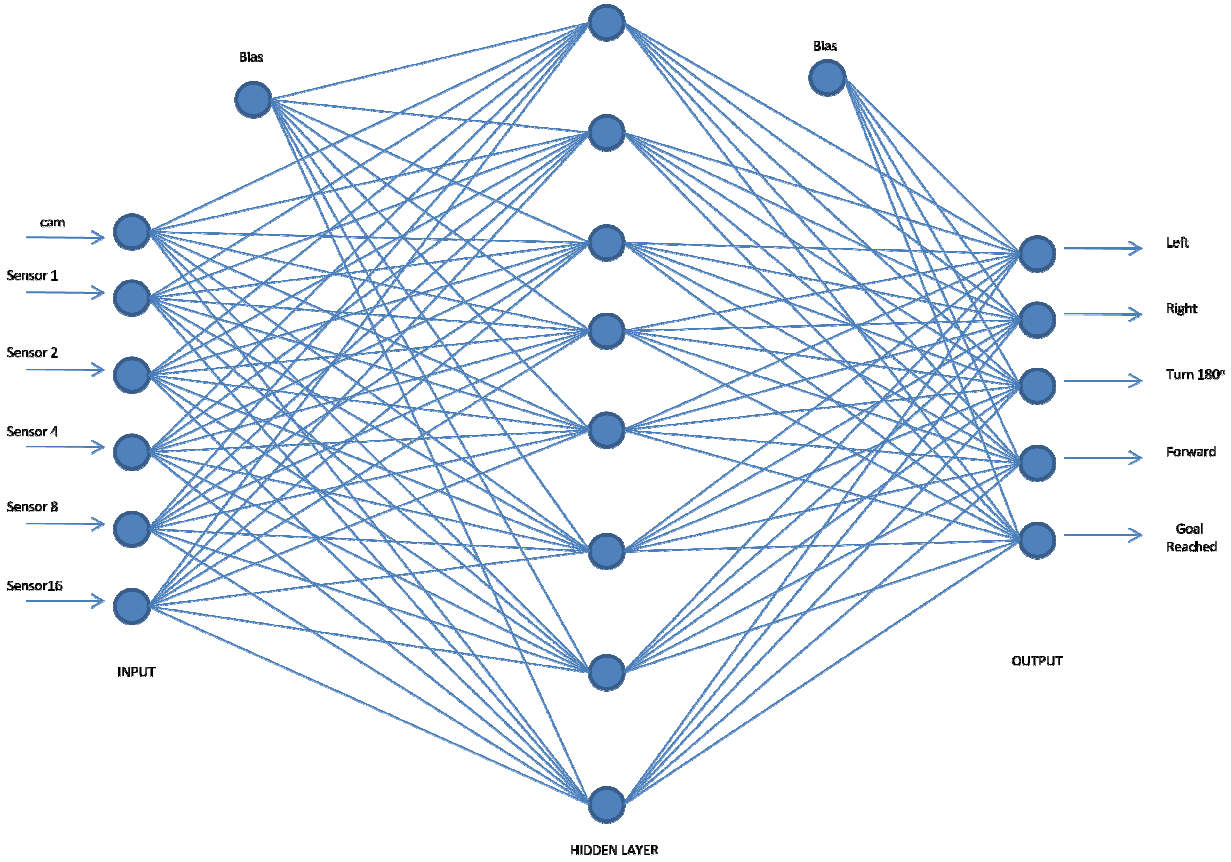

Figure 4.30 FF-MLP ANN

Training is performed using a bipolar function, since it converges more rapidly than a unipolar function, and does not get stuck in a local minimum. Furthermore, it has proven that using a single hidden layer is sufficient [Zur92]. A learning rate of 0.2 is more than enough to implement the learning and no momentum is needed during the weight updates.

The results for the FF-MLP are reported as follows. It can be demonstrated how well the results approximated the expected outcome by looking at the length of training, and the error after training: a square error of 0.04 in 54 epochs.

Original Weights

```
v[0][0] = -0.337800  v[0][1] = 0.277100  v[0][2] = 0.285900  v[0][3] = -0.332900
v[0][4] = -0.337800  v[0][5] = 0.277100  v[0][6] = 0.285900  v[0][7] = 0.000000
v[1][0] = 0.197000   v[1][1] = 0.319100  v[1][2] = -0.144800 v[1][3] = 0.359400
v[1][4] = -0.337800  v[1][5] = 0.277100  v[1][6] = 0.285900  v[1][7] = 0.000000
v[2][0] = 0.309900   v[2][1] = 0.190400  v[2][2] = -0.034700 v[2][3] = -0.486100
v[2][4] = -0.337800  v[2][5] = 0.277100  v[2][6] = 0.285900  v[2][7] = 0.000000
v[3][0] = -0.337800  v[3][1] = 0.277100  v[3][2] = 0.285900  v[3][3] = -0.332900
v[3][4] = -0.337800  v[3][5] = 0.277100  v[3][6] = 0.285900  v[3][7] = 0.000000
v[4][0] = 0.197000   v[4][1] = 0.319100  v[4][2] = -0.144800 v[4][3] = 0.359400
v[4][4] = -0.337800  v[4][5] = 0.277100  v[4][6] = 0.285900  v[4][7] = 0.000000
v[5][0] = 0.197000   v[5][1] = 0.319100  v[5][2] = -0.144800 v[5][3] = 0.359400
v[5][4] = -0.337800  v[5][5] = 0.277100  v[5][6] = 0.285900  v[5][7] = 0.000000
v[6][0] = -0.337800  v[6][1] = 0.277100  v[6][2] = 0.285900  v[6][3] = -0.332900
v[6][4] = -0.337800  v[6][5] = 0.277100  v[6][6] = 0.285900  v[6][7] = 0.000000

w[0][0] = -0.140100  w[0][1] = -0.337800  w[0][2] = 0.277100  w[0][3] = 0.285900
w[0][4] = -0.332900
w[1][0] = 0.491900   w[1][1] = 0.197000   w[1][2] = 0.319100  w[1][3] = -0.144800
w[1][4] = 0.359400
w[2][0] = -0.291300  w[2][1] = 0.309900  w[2][2] = 0.190400   w[2][3] = -0.034700
w[2][4] = -0.486100
w[3][0] = -0.397900  w[3][1] = -0.337800  w[3][2] = 0.277100  w[3][3] = 0.285900
w[3][4] = -0.332900
w[4][0] = 0.358100   w[4][1] = 0.197000   w[4][2] = 0.319100  w[4][3] = -0.144800
w[4][4] = 0.359400
w[5][0] = -0.140100  w[5][1] = -0.486100  w[5][2] = -0.337800 w[5][3] = 0.277100
w[5][4] = 0.285900
w[6][0] = 0.491900   w[6][1] = 0.197000   w[6][2] = 0.319100  w[6][3] = -0.144800
w[6][4] = 0.359400
w[7][0] = -0.291300  w[7][1] = -0.337800  w[7][2] = 0.277100  w[7][3] = 0.285900
w[7][4] = -0.332900
w[8][0] = -0.397900  w[8][1] = -0.486100  w[8][2] = -0.337800 w[8][3] = 0.277100
w[8][4] = 0.285900
```

EPOCH 0 results
SQUARED ERROR = 4.412942

```
yk[0][0] = -0.193476   yk[0][1] = -0.414966   yk[0][2] = -0.262091
yk[0][3] = 0.241975    yk[0][4] = -0.012578
yk[1][0] = -0.150649   yk[1][1] = -0.150792   yk[1][2] = 0.209086
yk[1][3] = 0.170916    yk[1][4] = -0.286343
yk[2][0] = -0.182772   yk[2][1] = -0.355603   yk[2][2] = -0.104967
yk[2][3] = 0.276499    yk[2][4] = -0.000805
yk[3][0] = 0.067572    yk[3][1] = 0.099984    yk[3][2] = 0.465428
yk[3][3] = 0.050788    yk[3][4] = -0.203721
yk[4][0] = -0.359316   yk[4][1] = -0.318714   yk[4][2] = 0.123400
yk[4][3] = 0.207205    yk[4][4] = -0.362728
yk[5][0] = -0.457857   yk[5][1] = -0.337391   yk[5][2] = -0.020533
yk[5][3] = 0.210286    yk[5][4] = -0.466952
yk[6][0] = -0.368017   yk[6][1] = -0.221686   yk[6][2] = 0.039145
yk[6][3] = 0.005920    yk[6][4] = -0.504124
yk[7][0] = -0.353469   yk[7][1] = -0.291706   yk[7][2] = -0.161403
yk[7][3] = 0.042528    yk[7][4] = -0.461811
```

EPOCH 54 results
SQUARED ERROR = 0.049036

```
yk[0][0] = -0.854167   yk[0][1] = -0.843023   yk[0][2] = -0.999479
yk[0][3] = 0.922547    yk[0][4] = -0.944988
yk[1][0] = -0.907736   yk[1][1] = -0.943364   yk[1][2] = -0.911109
yk[1][3] = 0.940624    yk[1][4] = 0.805729
yk[2][0] = -0.974795   yk[2][1] = -0.966514   yk[2][2] = -0.989919
yk[2][3] = 0.916675    yk[2][4] = -0.822502
yk[3][0] = -0.806143   yk[3][1] = -0.813300   yk[3][2] = 0.832748
yk[3][3] = -0.984543   yk[3][4] = -0.929286
yk[4][0] = -0.987002   yk[4][1] = 0.846088    yk[4][2] = -0.955212
yk[4][3] = -0.934091   yk[4][4] = -0.995401
yk[5][0] = 0.879271    yk[5][1] = -0.975997   yk[5][2] = -0.890687
yk[5][3] = -0.927014   yk[5][4] = -0.963382
yk[6][0] = -0.990044   yk[6][1] = 0.935454    yk[6][2] = -0.863333
yk[6][3] = -0.905766   yk[6][4] = -0.956067
yk[7][0] = 0.881596    yk[7][1] = -0.975783   yk[7][2] = -0.892371
yk[7][3] = -0.927638   yk[7][4] = -0.963512
```

Final Weights

```
v[0][0] = -0.880515  v[0][1] = 0.147550  v[0][2] = 0.660321  v[0][3] = -0.622061
v[0][4] = -0.384115  v[0][5] = -0.089316 v[0][6] = -0.069400 v[0][7] = 0.326917
v[1][0] = 0.807159   v[1][1] = -0.591215 v[1][2] = 0.238719  v[1][3] = 0.567324
v[1][4] = 0.095290   v[1][5] = 0.893257  v[1][6] = 1.051813  v[1][7] = 0.403267
v[2][0] = 1.332556   v[2][1] = -0.279800 v[2][2] = 0.284170  v[2][3] = -0.125058
v[2][4] = -0.676883  v[2][5] = 1.137538  v[2][6] = 0.883645  v[2][7] = -0.230856
v[3][0] = -0.103470  v[3][1] = 0.319336  v[3][2] = -0.134176 v[3][3] = -0.221423
v[3][4] = -1.485696  v[3][5] = 0.548287  v[3][6] = -0.680959 v[3][7] = -1.282912
v[4][0] = 0.492612   v[4][1] = 1.600550  v[4][2] = -0.197453 v[4][3] = 1.042411
v[4][4] = -1.009428  v[4][5] = -0.227811 v[4][6] = 0.483987  v[4][7] = -0.262954
v[5][0] = 0.080115   v[5][1] = 1.160435  v[5][2] = -0.132804 v[5][3] = 0.889292
v[5][4] = -0.237255  v[5][5] = -0.472092 v[5][6] = 0.652154  v[5][7] = 0.371169
v[6][0] = 0.222185   v[6][1] = -0.511007 v[6][2] = -0.901713 v[6][3] = 1.530660
v[6][4] = 0.443041   v[6][5] = 0.985379  v[6][6] = 0.192891  v[6][7] = 0.316051
```

```
w[0][0] = -1.662523  w[0][1] = -1.840577  w[0][2] = -1.778386  w[0][3] = -0.788332
w[0][4] = -1.369595
w[1][0] = 1.174209   w[1][1] = -0.854926  w[1][2] = 1.883831   w[1][3] = -0.216396
w[1][4] = 0.415857
w[2][0] = -2.125350  w[2][1] = 1.156461   w[2][2] = 1.367748   w[2][3] = -0.971803
w[2][4] = -0.700862
w[3][0] = -0.456096  w[3][1] = -0.953333  w[3][2] = -0.226203  w[3][3] = 0.184698
w[3][4] = -0.854488
w[4][0] = -0.804198  w[4][1] = 1.116394   w[4][2] = 1.262802   w[4][3] = 0.438196
w[4][4] = 2.169739
w[5][0] = -0.237620  w[5][1] = -0.892030  w[5][2] = -0.485529  w[5][3] = 2.176842
w[5][4] = 1.002818
w[6][0] = 1.651614   w[6][1] = -0.894523  w[6][2] = 0.928502   w[6][3] = -1.055861
w[6][4] = 0.937385
w[7][0] = -0.516206  w[7][1] = -1.545429  w[7][2] = 1.226545   w[7][3] = -0.073351
w[7][4] = 0.146485
w[8][0] = -0.874272  w[8][1] = -0.847023  w[8][2] = -0.284055  w[8][3] = 1.342792
w[8][4] = 1.091902
```

Chapter 5. Conclusions and Future Work

5.1 CONCLUSION

The use of classical logic algorithms and artificial neural networks offers a good solution to the problem of obstacle-avoidance while negotiating a maze environment. It was proven that by using both approaches, both of them produced identical results. The two approaches were compared in the amount of turns made; giving as a result that both techniques used the same amount of movements to negotiate the maze. Although both techniques got to the same results, the implementation can vary in terms of costs and time of performance. Classical logic would be implemented using a microcontroller, whereas ANN can be implemented in a field-programmable gate array (FPGA) having a great impact in reducing the manufacture cost. In terms of performance, classical logic system performs sequentially throughout the algorithm; whereas, because of its massive parallel processing, ANN (once trained) can reduce the time of reaction significantly towards the possible scenarios that the robot is exposed to.

It can be concluded that the Feed-Forward Multi-Layered Perceptron proposed in this thesis, can be used as a function approximator of any type e.g. a sine function. Having said the latter, it can be concluded that this type of architecture can in fact be implemented in any physical machine. However, other factors affecting the performance of a real robot must be considered as well. Such considerations include slippery surfaces, velocity compensation of individual tires, correct calibration of individual sensors, etc.

5.2 FUTURE WORK

There are several things that can be done as future work. One of them was implemented in parallel with this thesis work, which is a prototype implementation of a physical autonomous vehicle (PAV). The prototype is composed of the following:

- five proximity sensors
- one HCS12 microcontroller
- one CMOS camera
- two 12V/18 Amp-Hr Lead-Acid Batteries
- two wheel encoders
- two LM18200 motor drivers
- one power distribution board

The prototype can be used to carry out the physical implementation of what has been discussed in this thesis with very promising results. However, some design and/or implementation of both software and hardware implementation can be revised in order to enhance the performance of the PAV.

References

[Sho97] Anil Nerode, “Logic for Applications” Springer-Verlag, New York, 1997.

[Lin96] Chin-Teng Lin, “Neural Fuzzy Systems:A Neuro-Fuzzy Synergism to Intelligent Systems” Prentice Hall, 1996.

[Nau97] Detlef Nauck, “Foundations of Neuro-Fuzzy Systems” John Wiley & Sons, England, 1997.

[Fau94] L. Fausett. “Fundamentals of Neural Networks.” Prentice Hall, Upper Saddle River, NJ 1994.

[Zur92] J. M. Zurada. “Introduction to Artificial Neural Systems.” West Publishing Company,St. Paul, MN 1992

[Yen99] John Yen, “Fuzzy Logic: Intelligence, control and information” Prentice Hall, New Jersey, 1999.

[Dor98] Richard C. Dorf, “Modern Control Systems” Addison Wesley Longman, Inc. Menlo Park, CA. 1998.

[Kot97] Ronald Kotulak, “Inside the Brain. Revolutionary discoveries of how the mind works”, Andrews McMeel Publishing, 1997

[Sch97] R. Schalkoff. “Artificial Neural Networks.” McGraw-Hill, New York, NY 1997.

[Sam07] Sandhya Samarasinghe, “Neural Networks for applied sciences and engineering.” Taylor & Francis Group, Florida, 2007.

[Siv07] S.N. Sivanandam, “Introduction to Fuzzy Logic using MATLAB” Springer. Tamil, Nadu India. 2007

[Kar96] Stamatios V. Kartalopoulos, “Understanding neural networks and fuzzy logic: Basic concepts and Applications.” The institute of Electrical and Electronics Engineers, Inc., New York, 1996.

[Ros05] Timothy J. Ross, “Fuzzy Logic with Engineering Applications,”John Wiley & Sons, England, 2005.

[Cho07] Tommy W S Chow, “Neural Networks and Computing: Learning algorithms and applications.” Imperial College Press, London, 2007.

Appendix

Neural Network Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

// DEFINE VARIABLES & CONSTANTS

#define     INN 6
#define     HIDN 8
#define     OUTN 5
#define     ROWT 8
#define     ROWV INN+1
#define     ROWW HIDN+1
#define     TRAIN_S  8
#define     TEST  1

int         epoch,
            stopc;
      //                                  bias x1  x2  x3   x4 x5  cam
int         in_v_b[ROWT][ROWV] ={  {1,  -1, -1, -1, -1, -1, -1},
                                   {1,   1, -1, -1, -1,  1,  1},
                                   {1,   1, -1, -1, -1,  1, -1},
                                   {1,   1,  1,  1,  1,  1, -1},
                                   {1,  -1, -1,  1,  1,  1, -1},
                                   {1,   1,  1,  1, -1, -1, -1},
                                   {1,  -1, -1,  1,  1,  1,  0},
                                   {1,   1,  1,  1, -1, -1, -1},
                                   {1,   1, -1, -1, -1,  1, -1},
                                   {1,   1, -1, -1, -1,  1,  1}},


      //                                      left    right  180   fwd    GR
            target_vb[ROWT][OUTN]={
                                          { -1,    -1,     -1       1,    -1},
                                          { -1,    -1,     -1,      1,     1},
                                          { -1,    -1,     -1,      1,    -1},
                                          { -1,    -1,      1,     -1,    -1},
                                          { -1,     1,     -1,     -1,    -1},
                                          {  1,    -1,     -1,     -1,    -1},
                                          { -1,     1,     -1,     -1,    -1},
                                          {  1,    -1,     -1,     -1,    -1},
                                          { -1,    -1,     -1,      1,    -1},
```

```
                                                { -1,    -1,    -1,      1,    1}};


        double          v[ROWV][HIDN] = {
                        {-0.3378,  0.2771,  0.2859, -0.3329, -0.3378,  0.2771,  0.2859},
                        { 0.1970,  0.3191, -0.1448,  0.3594, -0.3378,  0.2771,  0.2859},
                        { 0.3099,  0.1904, -0.0347, -0.4861, -0.3378,  0.2771,  0.2859},
                        {-0.3378,  0.2771,  0.2859, -0.3329, -0.3378,  0.2771,  0.2859},
                        { 0.1970,  0.3191, -0.1448,  0.3594, -0.3378,  0.2771,  0.2859},
                        { 0.1970,  0.3191, -0.1448,  0.3594, -0.3378,  0.2771,  0.2859},
                        {-0.3378,  0.2771,  0.2859, -0.3329, -0.3378,  0.2771,  0.2859},
                                                },


w[ROWW][OUTN] = {
                       {-0.1401, -0.3378, 0.2771,  0.2859, -0.3329},
                        { 0.4919, 0.1970,  0.3191, -0.1448,  0.3594},
                        {-0.2913, 0.3099,  0.1904, -0.0347, -0.4861},
                        {-0.3979, -0.3378, 0.2771,  0.2859, -0.3329},
                        { 0.3581, 0.1970,  0.3191, -0.1448,  0.3594},
                        {-0.1401, -0.4861,-0.3378,  0.2771,  0.2859},
                        { 0.4919, 0.1970,  0.3191, -0.1448,  0.3594},
                        {-0.2913, -0.3378, 0.2771,  0.2859, -0.3329},
                        {-0.3979, -0.4861,-0.3378,  0.2771,  0.2859}},

z_inj[HIDN],
zj[ROWW],

y_ink[OUTN],
yk[TRAIN_S][OUTN],

delta_k[ROWW],
delta_wjk[ROWW][OUTN],
delta_j[ROWV],
delta_inj[ROWV],
delta_vij[ROWV][HIDN],

sqerr,
trerr    = 0.05,
lr       = 0.2;

double          temp, temp1, temp2;
int             testa;
double actfun(double);
void print_first();
void print_term();
void print_sec();
void print_train();
```

```
void print_train2();
void print_corv();
void print_up();
void print_getd();

// MAIN PROGRAM
void main(){
srand((unsigned int)time((time_t *)NULL));

if ( TEST == 0 )        stopc = 2;
else                    stopc =3000;


int i,j,k,l, pass, cs;

printf("Original Weigths\n");
        for (i = 0; i < ROWV; i++){
                for (j = 0; j < HIDN; j++){
                        temp = v[i][j] ;
                        printf (" v[%d][%d] = %f ", i,j,temp);
                }
                printf("\n");
        }

        printf("\n");

        for (i = 0; i < ROWW; i++){
                for (j = 0; j < OUTN; j++){
                        temp = w[i][j];
                        printf (" w[%d][%d] = %f ", i,j,temp);
                }
                printf("\n");
        }



        // Starting algorithm of Backpropagation
        for (epoch = 0; epoch < stopc; epoch++ ){
        sqerr=0;
                // Pass the Input vector to the net
                for (pass=0; pass<TRAIN_S; pass++){

        //  Multiply first layer
        // Perform: z_inj i+= Xi * Vij
        if (TEST == 0) print_first();
        for (i=0; i<HIDN; i++){ z_inj[i] = 0; }

        for (i=0; i<ROWV; i++){
```

```
                for (j=0; j<HIDN; j++){
                if (TEST == 0 ) printf ("z_inj[%d] = %f * %d + %f = ",j,v[i][j], in_v_b[pass][i], z_inj[j]);
                                z_inj[j] += v[i][j]*in_v_b[pass][i];
                                if (TEST == 0 ) printf (" %f\t", z_inj[j]);
                        }
                        if (TEST == 0 ) printf("\n");
                }

// Activate Neurons: zj
                zj[0] = 1 ;
                if (TEST == 0 ) printf ("\t\t\tactf [0] = %f\n", zj[0]);
                for (i=0; i<HIDN; i++){
                        if (TEST == 0 ) printf ("ACC[%d]= %f\t", i, z_inj[i]);
                        zj[i+1]= actfun(z_inj[i]);
                        if (TEST == 0 ) printf ("actf [%d] = %f \n", i+1, zj[i+1]);
                }
                if (TEST == 0) print_term();
                if (TEST == 0 ) printf("\n");
// Perform: y_ink i+= zj * Wij
                if (TEST == 0) print_sec();
                for (i=0; i<OUTN; i++){ y_ink[i] = 0; }

                for (i=0; i<OUTN; i++){
                        for (j=0; j<ROWW; j++){
                                y_ink[i] += zj[j] * w[j][i];
                                if (TEST == 0 ) printf (" %f * %f = %f \n", zj[j], w[j][i], y_ink[i]);
                        }
                if (TEST == 0 ) printf("\n");
                }
// Activate Neurons: yk

                for (i=0; i<OUTN; i++){
                        if (TEST == 0 ) printf ("ACC[%d]= %f\t", i, y_ink[i]);
                        yk[pass][i] = actfun(y_ink[i]);
                        if (TEST == 0 ) printf ("actf [%d] = %f", i, yk[pass][i]);

                        if ( TEST == 0 ) {
                                if ( yk[pass][i]<0.5 ) printf (" Output    0\n");
                                else                           printf (" Output    1\n");
                        }

                        sqerr += pow((target_vb[pass][i]-yk[pass][i]),2);

                        if ( TEST == 0 ) printf ("Pass error %f\n", sqerr);
                }
                if (TEST == 0) print_term();
                if ( TEST == 0 ) printf("\n");
//  Calculate delta_k, delta_wjk
```

```
                        if (TEST == 0) print_train();
                        if (TEST == 0) print_train2();
for (i=0; i<OUTN; i++){
        delta_k[i] = (target_vb[pass][i] - yk[pass][i])*(1+yk[pass][i])*(1-yk[pass][i])*0.5;

                if ( TEST == 0 ) printf ("Delta_k[%d] = %f\n", i, delta_k[i]);
                        }
                        if (TEST == 0) print_term();
                        if ( TEST == 0 ) printf("\n\n");
// Correction term for W, delta_wjk
        for (i=0; i<ROWW; i++){ delta_inj[i] = 0; }

                if (TEST == 0) print_getd();
                        for (i=0; i<ROWW; i++){
                                for (j=0; j<OUTN; j++){
                                        if ( i==0 ) delta_wjk[i][j] = lr * delta_k[j] ;
                                        else delta_wjk[i][j] = lr * delta_k[j] * zj[i];

        if ( TEST == 0 ) {
        if ( i == 0 ) printf ("delta_wjk[%d][%d] =  %f * %f = %f\t", i,j,lr, delta_k[j] , delta_wjk[i][j]);
        else printf ("delta_wjk[%d][%d] =  %f * %f * %f= %f\t", i,j,lr, delta_k[j] , zj[i],delta_wjk[i][j]);
        }

        if ( i>0 ){
        delta_inj[i] += delta_k[j] * w[i][j];
        if ( TEST == 0 ) printf (" delta_inj[%d] = %f * %f = %f", i, delta_k[j] , w[i][j], delta_inj[i]);
                }
        if ( TEST == 0 ) printf ("\n");
                }
        if ( i>0 ) { delta_j[i] = delta_inj[i]*((1+zj[i])*(1-zj[i])*0.5 ); }
                }
        if ( TEST == 0 ) printf("\n");

        for (i=1; i<ROWW; i++){ if ( TEST == 0 ) printf ("delta_j[%d] = %f \n", i, delta_j[i]); }

        if (TEST == 0) print_term();
        if ( TEST == 0 ) printf("\n");
//  Correction term for V, delta vij

if (TEST == 0) print_corv();
   for (i=0; i<ROWV; i++){
        for ( j=0; j<HIDN; j++){
          delta_vij[i][j]= lr*delta_j[j+1]*in_v_b[pass][i];
if ( TEST == 0 ) printf ("delta_vij[%d][%d] = %f * %f * %d = %f\t", i,j,lr,delta_j[j+1],in_v_b[pass][i],
delta_vij[i][j]);
          }
          if ( TEST == 0 ) printf("\n");
```

```
        }
        if (TEST == 0) print_term();
        if ( TEST == 0 ) printf("\n\n");
//  Update weights matrix W, delta wjk

if (TEST == 0) print_up();
        for (i=0; i<ROWW; i++){
                for (j=0; j<OUTN; j++){
        if ( TEST == 0 ) printf("w[%d][%d] = %f\t->\t%f + %f\t->", i,j, w[i][j], w[i][j], delta_wjk[i][j]);
                        w[i][j] +=delta_wjk[i][j];
        if ( TEST == 0 ) printf("\t%f\t\t", w[i][j]);
                }
        if ( TEST == 0 ) printf("\n");
                        }
// Update weight matrix V, delta vij
        if ( TEST == 0 ) printf("\n");
                for (i=0; i<ROWV; i++){
                        for (j=0; j<HIDN; j++){
                                if ( TEST == 0 ) printf("v[%d][%d] = %f\t->", i,j, v[i][j]);
                                        v[i][j] +=delta_vij[i][j];
                                        if ( TEST == 0 ) printf("\t%f\t\t", v[i][j]);
                }
                if ( TEST == 0 ) printf("\n");
                        }

                }

// Test SQUARE ERROR of net
sqerr = sqerr/TRAIN_S;
        if ( TEST == 0 || sqerr < 0.05 || epoch == 0 || epoch == stopc-1 ) {
                printf ("\nEPOCH  %d results\n", epoch);
                printf ("SQUARED ERROR = %f\n", sqerr);

                for (pass=0; pass<TRAIN_S; pass++){
                for (i=0; i<OUTN; i++){
                printf("yk[%d][%d] = %f\t", pass, i, yk[pass][i]);
                }
        printf ("\n");
                }
                }
if ( sqerr < 0.05)        break;
        if ( epoch == 0 ) print_term();
// Final Results
        printf("\nFinal Weigths\n");
        for (i = 0; i < ROWV; i++){
                for (j = 0; j < HIDN; j++){
                        printf (" v[%d][%d] = %f ", i,j,v[i][j]);
                }
```

```
        printf("\n");
}
printf("\n");

for (i = 0; i < ROWW; i++){
        for (j = 0; j < OUTN; j++){
                printf (" w[%d][%d] = %f ", i,j,w[i][j]);
        }
        printf("\n");
}
}

// Activation function
double actfun(double opera){
        double act;
        act =   (2/(1+exp(-1*opera)))-1;
return act;
}
```

Classical Logic Code

```
MainProgram:

   gosub Mazecreation
   gosub Start

End
//*************************************************************************
//                    Subroutine to Plot the Maze
// Note: This subroutine includes another subroutine named Cell which
/       is the one to draw the rectangles forming a cell of 5x4
//*************************************************************************


Mazecreation:
 ClearScr              // clears the screen
 LineWidth 3           // this sets the width of the lines for the block cells
 // draw basic blocks


 y = 60                // y coordinate of where to start the block cell
 for j=0 to 3          // this will set 4 cells vertically
  x = 80               // x coordinate of where to start the block cell
   for i=0 to 4        //this will set 5 cells horizontally

     gosub Cell

   //This part is in charge of giving the series of the cell every 119 pixels
      x=x+119
   next
      y=y+119
 next

 //

 gotoxy 678,536    // This will go the start of the last box where the goal is
 lineto 740,536    //This traces the vertical line
 lineto 740,362    //This traces the horizontal line
 SetColor red      // This will be the color of the goal
 lineto 678,362    // sets the coordinate of the goal line


  SetColor white
 LineWidth 4
 // open doors between blocks
```

```
   y=60

  for j=0 to 3
    x= 200
    for i=0 to 4

      nn=random(10)
      if (nn<5 and i<4) or j=3
        gotoxy x,y+60
        n=random(10)
        if n<5 then lineto x,y+3
        if n>=5 then lineto x,y+117
      else
          gotoxy x-60,y+120
          n=random(10)
          if n<5 then lineto x-120,y+120
          if n>=5 then lineto x,y+120
      endif


      x=x+119
    next
    y=y+119
  next

  gotoxy 80,60      // this sets the entrance always in coordinates (80,60)
  lineto 80,120     //this is the entrance opening
  rLocate 48,90    // set the robot at the entrance of the maze
  rTurn 90         // it places the robot heading at 90 degrees
Return
//=============================================================
Cell:
  // draw 4 cell piece at x,y
  rectangle x,y,x+120,y+120

  // draw a random line
  gotoxy x+60,y+60
  n=Random (4)
  if n=0 then lineto x+60,y
  if n=1 then lineto x+60,y+120
  if n=2 then lineto x,y+60
  if n=3 then lineto x+120,y+60
Return
```

```
//*****************************************************************************
//*****************************************************************************
//*                  Subroutine to Negotiate the Maze
//*
//*****************************************************************************

Start:
  rSpeed 7
  rForward 60             // it will entrance the maze going forward 60 pixels


  while rLook() <> Red     //while not seeing the color red, keeo going

    // determine if current cell has more than one exit
    numExits = 0
    dir = 0          // dir holds a value of the direction of possible exits in the current cell.
    cnt = 0           //cnt holds the value of the possible exits


       if not(rFeel()& 16)

      cnt = cnt+1
      dir = 4 // left side
    endif


    if not(rFeel() & 4)

     cnt = cnt+1
      dir =dir+ 2 // straight ahead
    endif


       if not(rFeel() & 1)

      cnt = cnt+1
      dir = dir + 1 // right side
    endif


       // if there is only one exit, take it
      if cnt = 1
      if dir=4 then rTurn -90
      if dir=1 then  rTurn  90
      if dir=0 then rForward 60
     rForward 60
      continue
```

```
    endif

    if cnt = 0 // no exit, turn around and go back
       rTurn 180
       rForward 60
       continue
    endif


    // there are multiple exits to this cell
    //(start with left, then CW)
      if dir & 4
        rTurn -90
        rForward 60
      else

    // go for fwd in multiple exits
        rForward 60
      endif

  wend

Return
```

Curriculum Vita

Javier Alejandro Flores was born in Ciudad Juarez, Chihuahua, México on March 24, 1981. He is the youngest son of Javier Alejandro Flores and Virginia Garcia. He was raised during his first ten years of age in Delicias, Chihuahua, Mexico. Afterwards, he and his family moved to Ciudad Juarez, Chihuahua, Mexico where he lived there for another nine years graduating from "Escuela Preparatoria por Cooperacion El Chamizal" in the year 1999. He got accepted at the University of Texas at El Paso in 2000 where he pursued a Bachelor of Science in Electrical Engineering with concentration in Field and Devices, receiving his degree in December 2005. Furthermore, he appeared in the National Dean's List 2003-2004.

In 2006 he got accepted to continue his education towards a Master of Science in Electrical Engineering, where he focused his area of research in Artificial Neural Networks. He took classes related to CMOS digital design and VLSI design. While pursuing his degree he received the opportunity to serve as a teaching assistant for senior projects in electrical engineering for two years under the supervision of Professor Antonio Woo. He received his degree of Master of Science in Electrical Engineering in July 2009.

Permanent address: 2361 Tierra Humeda Dr.
El Paso, Texas, 79938

This thesis was typed by Javier Alejandro Flores.