Parallel Neural Network Recognition
A Multi-Agent System Approach

A. Cunha, C. Biscaia, M. Torres, L. Sobral & O. Belo

Departamento de Informática, Universidade do Minho,

4709 Braga Codex, Portugal

EMail: jls@di.uminho.pt

Abstract

This paper presents and discusses the design and the development of a pattern recognition agent based on neural networks. This agent is part of an intelligent navigation system, providing it with the necessary vision abilities so that it can be placed on a strange environment in order to explore and recognise its structures and specificities. Although similar, the properties of the recognised objects change through time and according to each specific environment. The flexibility required by such recognition process was implemented by several pattern recognition agents. Each agent is based on a neural network and can be trained on-line by a parallel training algorithm to allow an effective real time utilisation.

1 Introduction

Navigation systems have been acquired recently popularity on many applicational fields. A quick review on the software market reveals many attractive products proposals concerning such kind of systems. One may find navigation products for leisure, sports or tourism activities. Usually, a conventional navigation system uses a pre-defined model of the environment to support its navigation activities. This model usually restricts the focus of attention of such systems, which limits the range of environments were it effectively navigate. In order to provide the desired environment independence, we developed a system [1] were the navigation functions are isolated from the vision ones. The system is divided into a navigation and a vision modules composed of several agents. They cooperate with each other providing to the system the aptness and performance to explore, recognise and navigate over a strange environment, as an autonomous entity.

The vision module is organised in competence areas, each of them associated to a specific agent, to deal with the diversity of information that must be recognised. Most of these areas are concerned with the recognition of object classes that can be identified using pattern recognition technology. The problems raised in the recognition of such object classes are very similar. Due to these circumstances, a generic pattern recognition agent, based on neural networks, has been developed and integrated in the vision module.

Usually a neural network is trained once using a set of pattern example and, due to its generalisation property, we expect it to work well over a broader range of patterns. However, this is not always true. Specially, if we require that recognition remains effective through time and in different parts of the external environment. To deal with this problem the neural network must be retrained, every time a severe error occurs in a pattern identification. This allows the network to adapt itself to differences in the patterns being recognised and, additionally, to recover automatically from crashing situations. These mistakes can be detected because the knowledge acquired in previous navigation processes is stored in the external environment model maintained by the navigation module. The agents are able to compare this knowledge with the results obtained by its neural network and decide if a pattern was wrongly identified. Due to the slowness that characterises the training algorithms and due to the requirement of effective real-time utilisation, we implemented the training in a parallel machine to achieve a better performance.

This paper is related with the study, design and development of a generic pattern recognition agent, based on neural networks, that uses a continuous parallel training process to increase its effectiveness. Special attention is dedicated to the parallel implementation of the neural network training algorithm, to the decision process that rules the retraining and to the integration of the agent in the navigation system. A short description of the navigation agents and of the knowledge representation model is also presented, since they provide the basis for all the retraining processes.

2 The Pattern Recognition Agent

2.1 Overall Description

The vision module is composed of several pattern recognition agents. Each one is responsible to observe the external environment, recognise part of its objects, describe them and their positions. Being part of a navigation system, each recognition agent must provide reliable object descriptions that can be trusted by all other agents. This requirement lead us to complement a pattern recognition technique with the ability to adapt to variations on the patterns being recognised.

To achieve such effectiveness we used on-line retraining of a neural network. Whenever the agent detects that an object was wrongly identified, due to a variation in its properties, that object image is integrated in the training set of the network and its training is reinitiated. The detection of such misidentifications is possible because the navigation system maintains, systematically, a model of the external environment as the way to support its activities. This model contains knowledge acquired in previous navigation processes that can be used to validate the results of the neural network.

Basically, a recognition agent has two main activities:

- **Recognition**. Application of the incoming patterns to the neural network in order to obtain a first pattern identification. The neural network is retrained whenever the *intelligent unit* decides;
- **Dealing with Uncertainty**. If exist causes of uncertainty in the results of the neural network, they must be relieved before the description is delivered to the remaining agents. This is realised combining the neural network results, the knowledge acquired in previous navigation processes, and the internal rules related with the agent's competence area. If a pattern was wrongly identified and it is possible to the agent determine its correct identification, a feedback process occurs in order to retrain the neural network.

2.2 The Agent's Architecture

The agent tasks have very different computational requirements. The *recognition* task, since it is based on a neural network with on-line retraining, must be implemented in a parallel architecture in order to provide an effective real-time recognition. An efficient parallel programming language, appropriate for complex calculations and for interfacing with sensorial systems should be used. Furthermore, to implement the *dealing with uncertainty* task we need a language that can effectively implement inference schemes and communication with others agents. In order to satisfy these requirements, the pattern recognition agents were divided into functional units (see Figure 1):

- the **vision unit**, that is responsible for the recognition activities; implemented in a parallel machine (a Parsytec PowerExplorer), using parallel C and the Parix environment [5]; and
- the **intelligent unit**, that deals with the uncertainty; implemented in a common workstation, using Prolog [3] and Linda [4] in the implementation of the communication platform.

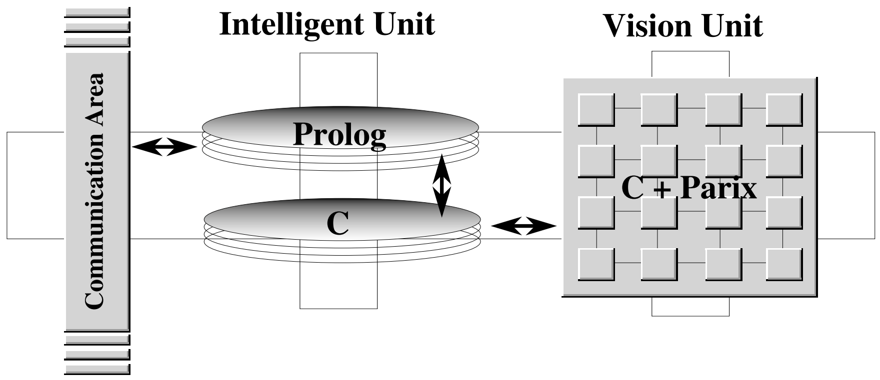

Figure 1: The Pattern Recognition Agent Architecture

2.3 The Parallel Learning Algorithm

The vision unit of the pattern recognition agent is a feedforward multilayer neural network. It consists of an input layer, with as many neurons as the number of pixels from the pattern image (a grey-scaled image), an hidden layer, and an output layer, were each neuron corresponds to one kind of pattern. The backpropagation algorithm was adopted as the learning algorithm. The patterns that compose the training set will be replaced in a FIFO manner during the retrain process: every time a wrong identification is made and the intelligent unit detects it, the corresponding pattern replaces the oldest one, in the training set, and the learning process is reinitiated from the point were it stoped.

The learning time of the neural network is usually excessive, specially in large networks, and when using large training data sets. Furthermore, the backpropagation algorithm has a rather slow rate of convergence [2]. The implementation of a such training algorithm on a parallel machine, decreases the learning time and allows a faster retrain of the neural network every time a wrong identification is detected. With a reduction of the training set, we could easily achieve the desired performance improvement. Although, it usually reduces the generalisation capability of the net, which is an undesirable consequence.

There are several manners of exploiting the parallelism in the implementation of the backpropagation learning algorithm [2], but we can distinguish two main approaches: the training parallelism and the spatial parallelism. The former implies the use of off-line weight updating and consists of dividing the training set into subsets of associated input-output patterns, which allows the training to be done in parallel. In the spatial parallelism, the computation associated to the neurons, in a given layer, is done in parallel. Although in the forward pass of the backpropagation algorithm, activities propagate sequentially through successive layers, they are done in parallel within a given layer. Similarly, the error signals propagated during the backward pass, as well as the weight updating, is done in parallel at each layer.

The chosen approach was to exploit the spatial parallelism of the algorithm, because it is more effective than the training parallelism when dealing with large networks, and when the training set is of medium sized. To exploit the spatial parallelism, in the backpropagation algorithm, the network was divided into slices, uniformly distributed among the available processing nodes (PN) (see Figure 2), being each one responsible for part of the neurons of each layer. Each processing node contains the connection weights between its neurons and the neurons of the next layer, the training data corresponding to its neurons in the input and output layers, and the outputs and the error signals for each one of its neurons.

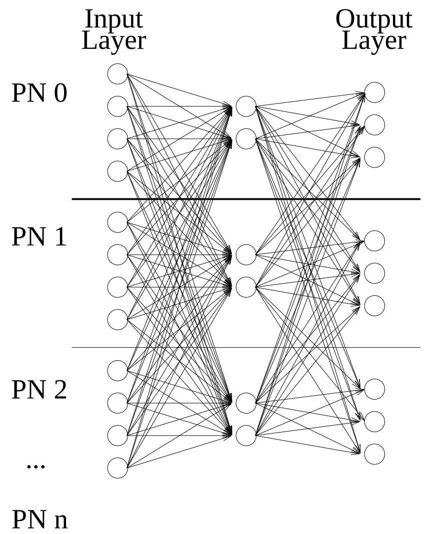

Figure 2: Spatial Parallelism in the Backpropagation Algorithm

During the forward pass of the backpropagation algorithm, each node computes the weighted averages of the outputs of the neurons that it contains, and sends the results to the corresponding neurons of the next layer. To reduce the number of messages a single message is composed for each node, containing all the information addressed to its neurons. In the backward pass each node sends the errors, computed at the output of each one of its neurons, to all other nodes in order to allow the on-line updating of the network weights. To improve efficiency, the message with the errors is transmitted using a broadcast mechanism.

The learning algorithm was implemented in a distributed memory parallel computer (a Parsytec PowerExplorer) were the computing nodes are physically interconnected in a mesh topology and interact with each other through message passing. Each processing node has two processes: a *worker process*, responsible for the computation itself, and a *router process* responsible for message deliverance (see Figure 3). The interconnection of the processing nodes is based on a virtual ring topology, because it easily implements broadcasting messages, frequently used in the learning algorithm. Although, different virtual topologies can be implemented simply by changing the router processes. One of the nodes contains a special process, the *system controller*, that controls the training process, and is responsible for the interface with the intelligent unit.

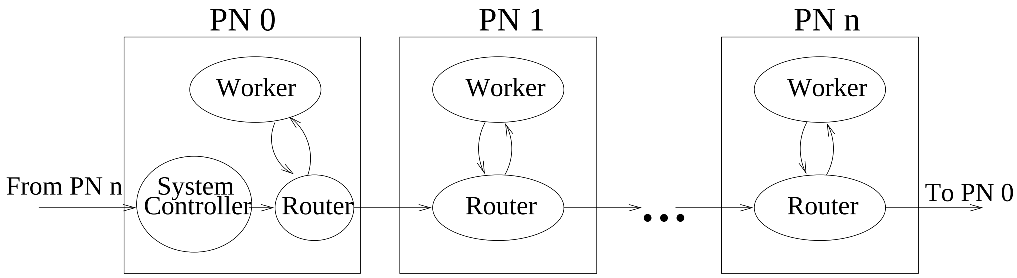

Figure 3: Computing Nodes Processes

The parallel machine is not only used to train the neural network but also in the recognition process. The algorithm for this process is identical to the forward pass of the parallel training algorithm. If the parallel machine is in a retraining process and the system controller receives an identification request, the retrain is suspended and the identification is done, using a copy of the previous network weights to the present retraining process.

3 The Intelligent Navigation System

3.1 Overall Description

The system follows a typical multi-agent systems architecture. All its units are agents with specific tasks allocated. Agents are reactive entities responding in a timely manner to events occurred in the system's environment. To provide environment independence, the system has a major division between navigation agents and vision agents that changes according to the desired system focus of attention.

The actual prototype system does not incorporate the means to control physical parts, which include all the items related with sensors and locomotion mechanisms. These functionalities are simulated through a special software agent that receives the orders, addressed to the mechanic components, and reports them, on a private system's monitor.

3.2 The System's Architecture

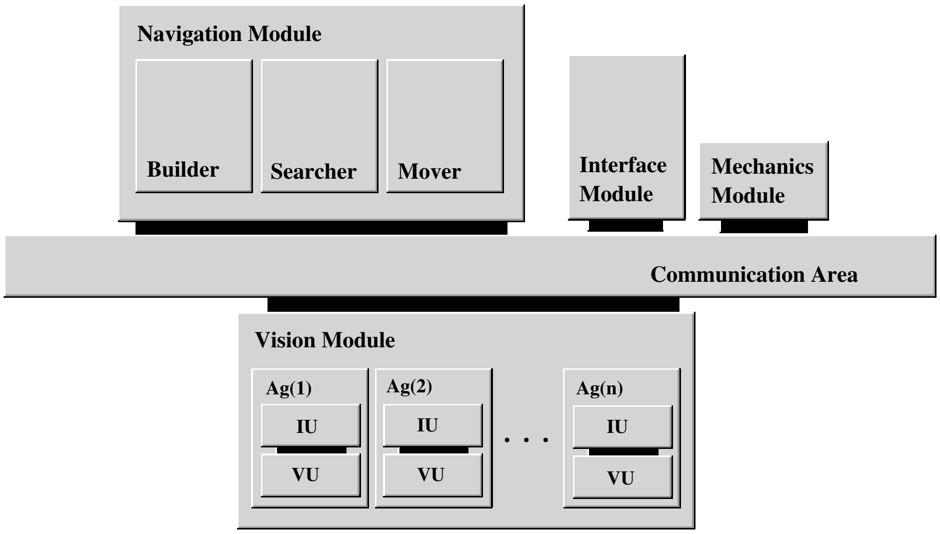

Figure 4: The System's Architecture

Basically, the system is constituted by four different modules:

- **Vision**, integrates the system's vision agents (including the presented pattern recognition agent) that extract the meaning about the vehicle's surrounding environment, based on incoming raw data and on the knowledge acquired in previous navigation processes; they define the system focus of attention;
- **Navigation**, composed by three different agents:
  - the **builder**, that is responsible to build, verify and maintain the environment's model according to the information sent by the vision module or, in some cases, by the interface module;
  - the **searcher**, that tries to find a specific path or place on the external environment, according to the instructions or commands received from the other modules, some defined utility functions or behaviour constraints; and
  - the **mover**, that is responsible for the local low-level navigation, in order to track the high-level paths produced by the searcher, and to rule the vehicle mechanics.
- **Interface**, is responsible to receive the user's commands and translate them into internal goals that the other system's modules try to

fulfill; it also behaves as a monitor, searching and collecting relevant knowledge from the communication area or data; and

- **Mechanics**, that executes the system's low level orders related with the physical system's devices control; this module was simulated through a special software agent.

The system's agents communicate with each other through message passing. To support such interaction process, it was developed a special data structure: the communication area. The communication area acts like a blackboard [6] [7] structure ensuring information exchange and storing, whenever necessary global system's knowledge. It is a fixed and dedicated structure as a way to reduce computational costs and to facilitate inter-agent communication. System's control is distributed over the agents, preventing the system against deadlock situations and avoiding contention.

3.3 The Knowledge Representation Model

Adequacy and expressiveness are two of the most important characteristics that a knowledge representation language must exhibit. These characteristics were critical in the development of the system's knowledge representation model. Collected and recognised by the system's vision units, the environment's descriptions demand an high level representation language. Such language must provide a semantic capable to express all the descriptions in an appropriate and effective way, so the system must be able to navigate intelligently. In order to provide the system with learning abilities, it must also be necessary to deal with "unknown" parts of the environment. Intelligent navigation is a time critical task, demanding fast and reliable path finding and internal structures updating every time the system detects new situations.

To achieve the referred features and objectives, the notion of scenery was introduced and used as the main element of the system's knowledge representation language. A scenery represents a particular view of an external environment's location. Different sceneries are generated according to the possible directions that the vehicle may assume when tries to reach a specific location. Each scenery is internally represented by a unique identification and an attribute list that characterise a location's view. Once introduced the scenery notion, the external environment can be mapped into an oriented graph, allowing for navigation through well defined algorithms. The requirement to mark unexplored parts of the environment is achieved by a special link attribute. The use of a list of attribute allows the adding of new attributes, maintaining compatibility with the previous structures.

Lets take a city example as the external environment where the system applies its navigation skills. We can verify that only a small part of the information associated with a local is independent of the direction followed by a vehicle when passing through it. Take, for instance, the information about the traffic signals. If all the information about one location is in the same scenery, the path finding algorithm will be slower, because of the need to identify which information is relevant in the followed direction. With this scenery notion there will be different sceneries to represent different directions in the same street, and others to represent the same cross-road according to the direction that the vehicle has followed to get to it. Figure 6 illustrates part of the view's scenery representation presented in Figure 5.

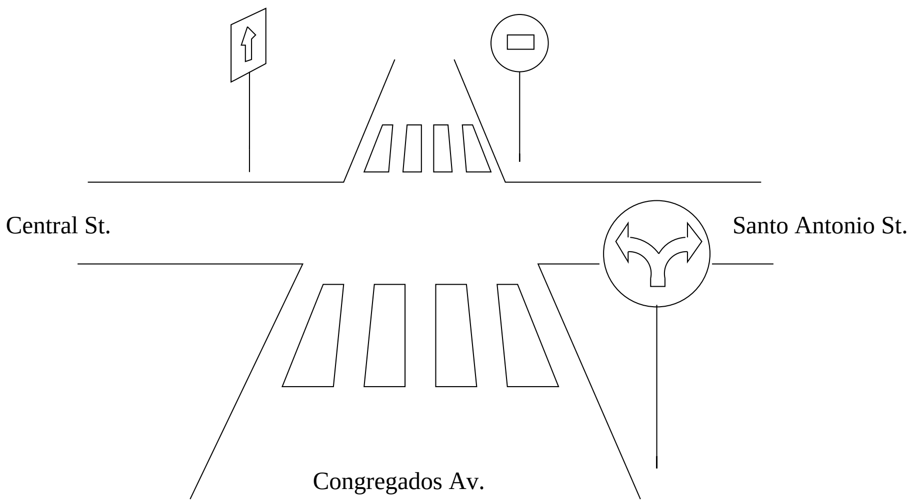

Figure 5: A City's Zone Representation

```
(...)
scenery(scenery_id(id1),[signal(obligation_to_turn_left_or_right),
        street_name(congregados_av),link(scenery_id(id2),meters(20),degrees(180))]).
scenery(scenery_id(id2),[zebra_cross,street_name(congregados_av),
        link(scenery_id(id3),meters(10),degrees(180))]).
scenery(scenery_id(id3),[cross_road,street_name(congregados_av),
        link(scenery_id(id4),meters(10),degrees(270)),link(scenery_id(id5),
        meters(10),degrees(180)),link(scenery_id(id6),meters(10),degrees(90))]).
scenery(scenery_id(id4),[signal(one_way_street),
        street_name(central_st),link(_,_,_)]).
scenery(scenery_id(id5),[zebra_cross,signal(forbidden_direction),
        street_name(congregados_av),link(_,_,_)]).
scenery(scenery_id(id6),street_name(santo_antonio_st),link(_,_,_)]).
(...)
```

Figure 6: A View's Scenery Representation

4 Conclusions and Future Work

This paper presented a parallel neural network recognition based agent, which is part of an intelligent navigation system. The navigation system has the ability to adapt itself to the changes of the environment, so the agent primary goal is a flexible and fast pattern training and recognition. Flexibility is achieved by an on-line retraining scheme. Fast training and recognition is achieved by a parallel neural network recognition and incremental parallel retraining. The independence between the vision and navigation modules achieved through the use of a blackboard based architecture allows for the changing of the system's focus of attention by simply modifying the agents in the vision module. However the prototype system reveals some particular situations that must be improved. Future research lines point to: improve overall system's security, efficiency and performance; development of the low-level navigation agents and the application of the system to real word applications.

Key Words

Intelligent Agents, Parallel Algorithms, Navigation Systems, Distributed Computer Systems.

References

[1] A.Cunha, C.Biscaia, M.Torres, L.Sobral, and O.Belo. Simulating the Use of Autonomous Intelligent Agents on Cellular Manufacturing Plant Floors. In *Proceedings of the 8th European Simulation Symposium*, Genoa, Italy, October 1996.

[2] I.Pitas, editor. *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks.* Jonh Wiley & Sons, 1993.

[3] M.Carlsson and J.Widen. *SICStus Prolog User's Manual.* Swedish Institute of Computer Science, January 1993.

[4] N.Carriero and D.Gelernter. Linda in context. *Communications of ACM*, 32(4):444–458, April 1989.

[5] Parsytec Computer GmbH. *Parix Manual*, December 1994.

[6] R.Engelmore and T.Morgan. *Blackboard Systems.* Addison-Wesley Publishing Company, Inc., 1988.

[7] V.Jagannathan, R.Dodhiawala, and L.Baum, editors. *Blackboard Architectures and Applications.* Academic Press, Inc, 1989.