An adaptive alternative for syntactic pattern recognition

Eduardo Rocha Costa, Andre Riyuiti Hirakawa, João José Neto

A. Adaptatives Techniques

Abstract— **Currently there are three basic types of pattern recognition, the syntactic, the statistical and the neural types. They all have advantages and disadvantages, some of which are troublesome. In statistical methods it is dificult to express structured information. Neural methods have problems to represent neural networks semantically. In syntactic methods it is lead to learn new rules. This is exactly the strong point in the method presented here, due to the inherent learning ability of adaptive automata. Our method not only solves the learning problem, but also realizes that it is a promising method since it shows many of the advantages of much more complex methods, such as self-organizing neural nets, which have adaptability.**

The method presented here is suitable to handle robot tracking applications where the goal is to find some object or position without needing details on information of the objects or the environment.

Basic geometric patterns, like rectangles and triangles are used here to ilustrate the recognition process using adaptive automata, and also to demonstrate the simplicity and efficiency of the method. This is a first step, from which it will be able to recognize a greater number and complex forms too.

Keywords— **Pattern Recognition, Adaptive automata, Robotics.**

I. Introduction

Pattern recognition is a process based on the capacity to distinguish different patterns and classify then according to their different attributes or different values assumed by a common attribute. Unfortunately, that isn't always an easy task.

As an example, let us study pictures in newspapers, consisting of black and white dots arranged in an array format. We may consider dots as basic characteristics or primitives. In order to try distinguishing different objects, in the picture, such as a ship and a car, we realize that the spatial arrays of these primitives are definitely different. To recognize different pictures we must apply an algorithm that takes into account the whole dot distribution. Depending on the object features and the purpose of the recognition, that process may be complex and hard. In order to solve that problem, the recognition process may include methods that simplify object features and reduces the information range. In this case, a set of primitives is defined and the original features are converted into particular primitives, as shown in the method described in this paper. The choice of the primitives in this set is a hard work that depends on object features and the main goals of the recognition process [1].

A. Pattern Recognition Process

The process pattern recognition comprehends information processing and transformation that start, form information gathered by sensors and performs adequate manipulation until results are obtained as the outputs of some classification algorithm.

• *Sensor* - Sensors are devices that gather information from the desired environment. Cameras, infrared sensors, ultrasonic sensors and so on usualy accomplish this task [2],[3].
• *Filtering and preprocessing* [2],[4] – This step is applied in order to simplify the final processing and recognition tasks. For example, a black-and-white image frame may be transformed into a binary logical array. Then, filtering and/or emphasizing methods may be applied in order to improving the information quality [5].
• *Feature Extraction Algorithm* - In this step the image is segmented and primitives are extracted. The initial information is analyzed to get the desired information and determine the primitives set. The original feature is compared with pre-defined subsets of patterns that will compose the final pattern to allow the recognition. The pattern subset is formed by primitives or vectors[1]. In this step, the adequate choice of the pattern makes the difference in the recognition process.
• *Classification* - It is the recognition process, where the primitive set or the feature vectors are analyzed to accomplish the desired identification. There are different classification algorithm for different applications:
– Syntactic. Uses graphs and grammars [5].
– Statistic. Uses Bayes networks or near neighbor methods [6].
– Neural. Uses different types of neural networks [6],[7].

B. Adaptive techniques

Any formal device may become adaptive, by adding and adptive mechanism to the device's usual formalism [8]. However, that does not add complexity to the original notation, which remains essentially untouched, except for the addition of the mechanisms responsible by dynamically modifying its behaviour. We will modify the simple formalism of the finite state automata, into adaptive finite automata, therefore increasing its power, making it equivalent to the Turing machine. This way, besides solving problems that before could not be solved by finite automata, these will incorporate an intrinsic learning power, which is a desirable effect for our purporse.

II. Concepts

A. Formalism

A.1 Automata

Finite Automata are devices whose formal definition is given below.
It is five tuple $(Q, \Sigma, \delta, q_0, F)$, with:
1. A finite, non empty set of states, denoted by Q
2. An finite, non empty alphabet, denoted by Σ
3. A transistion function, $\delta : (Q \times \Sigma) \rightarrow Q$
4. A initial state $q_0 \in Q$
5. A set of final states, $F \subseteq Q$

This type of automaton is called deterministic finite automaton, because given the current state and an input symbol, the destination state is uniquely defined. Any finite automaton recognizes some regular language. We can now illustrate a notation for this formalism with the following automaton.

$$M_1 = (\{a_1, b_1\}, \{\sigma_1, \sigma_2\}, \delta, q_1, \{a_1\}) \qquad (1)$$

TABLE I
Transistion Function.

q	σ	$\delta(q, \sigma) = q'$
a_1	σ_1	b_1
b_1	σ_2	a_1

A rule in this formalism has the general form $(q, \sigma) \rightarrow q'$ with $q, q' \in Q$ and $\sigma \in \Sigma$. Table I holds the two rules that define this automaton: $(a_1, \sigma_1) \rightarrow b_1$, and $(b_1, \sigma_2) \rightarrow a_1$. A graphical representation for this device is shown in Figure 1.

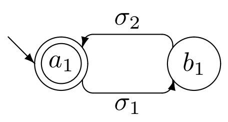

Fig. 1. Finite Automaton M_1.

A.2 Adaptive Automata

The formalism abovefor finite automata is simple. It may be converted into adaptive formalism by specifying the conditions under which it is expected to be modified, as well as the adaptations to take palce in each situation [8], [9].

An adaptive device has the form $AD = (ND_0, AM)$ where the device ND_0 is not adaptive, and AM is the associated adaptive mechanism.

The adaptive finite automaton is defined as in (2)

$$AA = (Q_0, AR_0, \Sigma, q_0, F, \mathcal{B}, \mathcal{A}) \qquad (2)$$

AR_0 is the set of adptive rules and is given in (3)

$$AR_0 \subseteq \mathcal{B} \times ((Q \times \Sigma) \times Q) \times \mathcal{A} \qquad (3)$$

where $(Q \times \Sigma) \times Q$ corresponds to δ in the formalism of the finite automaton. $\mathcal{B}$ and $\mathcal{A}$ are sets of adaptive actions, to be executed respectively *before* and *after* the transistion.

Each action is composed by, set of primitive adaptive actions as shown in the Table II.

TABLE II
Notation for primitive actions and rules.

Symbol	Meant
?[rule, action ϕ, action ψ]	Action of inspection
$-$[rule, action ϕ, action ψ]	Action of exclusion
$+$[rule, action ϕ, action ψ]	Action of inclusion

The format of rules r that define δ is the same of that in the underlying non adaptive formalism: ψ and ϕ are *before* and *after* adaptive actions, respectively $\phi \in \mathcal{B}$, $\psi \in \mathcal{A}$.

Table III shows the convetion adopted for denoting adaptive actions.

TABLE III
Graphical notation for adaptive actions.

Symbol	Nomenclature
$\Psi\bullet$	*before* adaptive action (ϕ)
$\bullet\Psi$	*after* adaptive action (ψ)

Notice, that in its general form Ψ is parametric, denoted $\psi(\alpha_1, \alpha_2, \ldots, \alpha_n)$ where α_i are arguments of the function Ψ.

Having more than one primitive adaptive action specified, all inspections and exclusions are performed first, and finally, the inclusions. Such composed actions are built from primitive actions.
As example, an adaptive automaton that accepts the language $\{w \mid w = (^n\ v\ )^n,\ n \geq 0\}$ is depicted in the Figure 2.

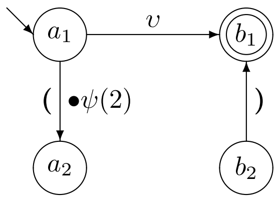

Fig. 2. Automaton for parenthesis levels.

In the Figure 2, $\bullet\psi(i)$ is the *after* adaptive action, and is described in the Table IV, noting that the new states a_{i+1}, b_{i+1}, that are not in the Figure 2 are automatically instanced every time that $\psi(i)$ is activated.

After the execution of transistion (a_1, ( ) $\rightarrow a_2$, $\bullet\psi(2)$, the aspect of the automaton becomes that on the Figure 3.

Notice that the transistion (a_1, ( ) $\rightarrow a_2$, $\bullet\psi(2)$ has been eliminated, and the transistions in (4), have been inserted.

$$\begin{array}{lcllll} (\ a_2, (\ ) & \rightarrow & a_3, & \mathcal{E}, & \bullet\psi(3) \\ (\ b_3,)\ ) & \rightarrow & b_2, & \mathcal{E}, & \mathcal{E} \\ (\ a_s, v\ ) & \rightarrow & b_2, & \mathcal{E}, & \mathcal{E} \end{array} \qquad (4)$$

TABLE IV
Description of Adaptive Action $\psi(i)$.

$\psi(i)$						
$-[$	$(\,a_{i-1}, (\,)$	$\rightarrow$	$a_i,$	$\mathcal{E},$	$\psi(i)$	$]$
$+[$	$(\,a_{i-1}, (\,)$	$\rightarrow$	$a_i,$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	$(\,a_i, \upsilon\,)$	$\rightarrow$	$b_i,$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	$(\,a_i, (\,)$	$\rightarrow$	$a_{i+1},$	$\mathcal{E},$	$\psi(i+1)$	$]$
$+[$	$(\,b_{i+1},)\,)$	$\rightarrow$	$b_i,$	$\mathcal{E},$	$\mathcal{E}$	$]$

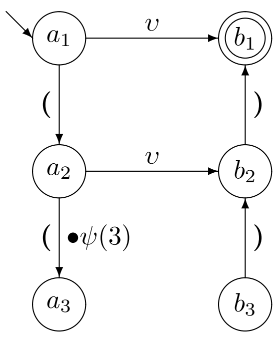

Fig. 3. Aspect of the adaptive automaton for parenteses levels after finding the first symbol (.

With such mechanism, adaptive finite automata become able to recognize more complex languages, for example $a^n b^n c^n$ or other context dependent languages. See section III-C.1.

B. Syntactic recognition of patterns

Syntactic methods may be classified as grammatical or graphical.

Graphical methods use graphs and trees, in their representation.

Grammatical methods use automata and parses of some types and the storage is in the form of grammars or automata.

Data entry in syntatic methods is primitive, and applies to basic forms, such as, straight lines and angles, or composite forms, e.g. squares and circles.

One may ask whether primitives are the best choice for pattern recognition. Human beings recognize their world in this way which encorage adopting such approach.

In order to illustrate this affirmation, we shall study, as example, the recognition of a dog. Several races exist, including half-breed, resulting from the crossing of races. If we meet a dog anywhere, no matter how would it seem, we are always sure that it is a dog, an not any other animal. Therefore, we can assume that we do not register images of each existing dog, but peculiar characteristics of this type of animal, independent of the race it belongs to. We can extend such reasoning to several other objects, such as chairs, trees, houses, etc.

In short, whichever objects are analyzed, we will be able to find peculiar characteristics, that enable us to classify similar objects, and objects of one same class, with small differences.

In the syntactic method there are many successfully applications, e.g. the recognition of cromosomes and Chinese characters recognition. We also point out here that the induction of context-free and context-dependent have been of little use because of their complexity.

III. Proposal

A. Usual implementation

After the image including the pattern to be recognized, is preprocessed and segmented, we will need a grammar already implemented and adjusted to the previously chosen primitives.

Therefore, for each project, we will need to choose new primitives before determining the grammar for the case. So, for each project all grammatical part needs to be created again. Hence, each project requires a separate study, which is time consuming because it's not a trivial task.

B. Adaptive automata replacing conventional ones

The use of adaptive automata in the solution of a specific problem allows the resolution of similar problems with only minor changes. It will adapt itself, reaching new configurations, so that it is capable to recognize another class of primitives for a different application. So using adaptive automata is a potentially time saving practice in such situations.

C. Specification of an Adaptive Automaton for the particular case of pattern recognition

C.1 Functionality

Here only the final process is described, since is well known and may be found in many texts on image processing, such as [2] and [3]. The input data to the process will be made through a matrix $n \times m$ that is restricted to 10×10 in this illustrative example. Figure 4 shows the graphical form of the data.

Fig. 4. Binarized image

$$\begin{matrix}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0\\
0&0&0&\mathbf{1}&0&0&\mathbf{1}&0&0&0\\
0&0&0&\mathbf{1}&0&0&\mathbf{1}&0&0&0\\
0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{matrix}$$

In figure 4, the 10x10 matrix of binarized points represents a square as the image to be recognized. After being trained the automaton must be capable to recognize that pattern represented by 1's.

First of all, a grid compatible with the application must be chosen, then the primitives are applied. In this case unitary grid size is chosen for best resolution. Figure 5 shows the primitives selected for this case.

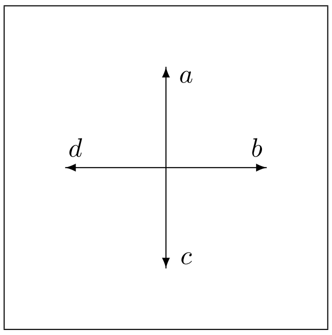

Fig. 5. Primitives for squares

The size of each primitive is unitary, and each pixel of the image will be a primitive in the corresponding direction. After doing so, the image in figure 4 is interpreted as being *dddaaabbbccc*. Afterwards, an adaptive automaton can be implemented, which will recognize squares and will have to recognize any shape of the form $a^n b^n c^n d^n$. This is a context dependent language, that may be recognized only by a device with the power of a Turing Machine [10]. Our proposed device has the same power of this formation, being able, therefore, to recognize languages with such complexity[9].

Fig. 6. Binarized equilateral triangle

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	**1**	**1**	0	0	0	0
0	0	0	**1**	0	**1**	0	0	0	0
0	0	**1**	**1**	**1**	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

D. Analysis of the Pattern recognition process

• Similarity is the main problem to be solved. Given a perfect square and another one with some noise, say, with a slightly defective edge, will the parser recognize these two patterns as members of the same class? This problem may be addressed in two ways: first, during the preprocessing of the image, when imperfections may be corrected, in order to minimize this effect; second, during the recognition phase, error repair techiniques may be used. This method is widely used in compilers. Both methods may be applied together, in the first phase, for minimizing image flaws. In the classification phase, error repair may be used to remove slightly deformations in the patterns.

• As noticed, in usual pattern recognition, scalability is a problem, since depending on the focus of the camera, the image of the same object will appear of different sizes, thus generating recognition errors. In the usual form, transformations are needed to be applied to the image, demanding extra processing time. In the present example, that is not important for the automaton, therefore it will recognize the form, independently of its scale.

• When an image is rotated, the recognizer cannot classify it as any of the known forms, because all primitives are also rotated by a certain angle. The solution, relatively simple, to solve this problem, is simply to rotate all the image that angle, before applying the extraction algorithms. Consequently, all the forms will be reduced to the same axle, thus not growing the complexity of the recognition process.

• Errors due to parallax may be solved by rotating the im-

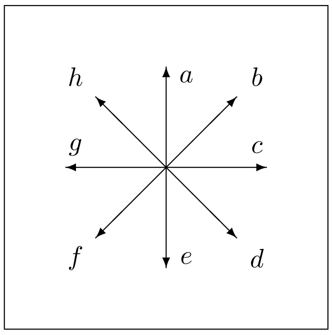

Fig. 7. Primitives for triangle

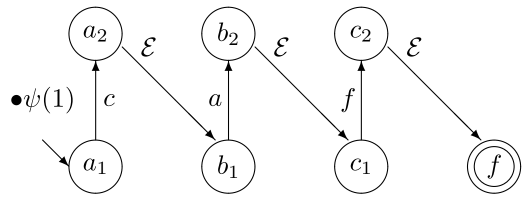

Fig. 8. Automaton recognition of triangles.

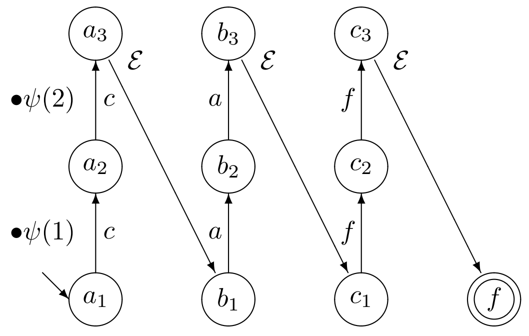

Fig. 9. Automaton recognition of triangles.

age around its z axle, i.e., the axle perpendicular to the lens of camera, before the phase of primitive extration. Texts on rotations of images can are found in the literature [11].

IV. Implementation

Figure 6 shows the graphical form of the input data, and is used to demonstrate how adaptive automata work. Figure 7 shows the primitives for this implementation. In the Figure, each primitive corresponds to two pixels in the Figure 6.

The string represented by this image is of the form $c^n a^n f^n$, actually $c^2 a^2 f^2$, whose recognizing automaton is graphically represented in Figure 8. After the adaptive action $\psi(i)$ is performed, the shape of the automaton will have the aspect shown in Figure 9. Finally, the automaton returns to the original configuration in Figure 8 again, and may be used for recognizing another string. The after-adaptive action $\psi(i)$ is defined in Table V.

TABLE V
Adaptive Action $\psi(i)$.

$\psi(i)$						
$-[$	$(a_i, \mathcal{E})$	$\rightarrow$	$b_{i-1},$	$\mathcal{E},$	$\mathcal{E}$	$]$
$-[$	$(b_i, \mathcal{E})$	$\rightarrow$	$c_{i-1},$	$\mathcal{E},$	$\mathcal{E}$	$]$
$-[$	$(c_i, \mathcal{E})$	$\rightarrow$	$f,$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	(a_i, c)	$\rightarrow$	$a_{i+1},$	$\mathcal{E},$	$\psi(i)$	$]$
$+[$	(b_i, a)	$\rightarrow$	$b_{i+1},$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	(c_i, f)	$\rightarrow$	$c_{i+1},$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	$(a_{i+1}, \mathcal{E})$	$\rightarrow$	$b_1,$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	$(b_{i+1}, \mathcal{E})$	$\rightarrow$	$c_1,$	$\mathcal{E},$	$\mathcal{E}$	$]$
$+[$	$(c_{i+1}, \mathcal{E})$	$\rightarrow$	$f,$	$\mathcal{E},$	$\mathcal{E}$	$]$

V. Conclusion

A new method for pattern recognition, based on adaptive automata was presented. From the limitations of usual syntactic methods, i. e., learning, flexibility, etc, our method extends it with use of the adaptive automata, which has an inherent learning ability, wich is highly convenient for pattern recognition. The new method seems promising. Its formalism is well-defined and simple, leading to very fast and simple implementations. The illustrating example show that the method works very well in the established conditions, and also, the method has low sensitivity to errors in the image, which is an inherent effect of the error-recovery syntactical technique used with automata. The implementation of the automata and the choice of the primitives change the efficiency of the method and must be analyzed carefully. The proposed method can be easily extended for recogniting complex forms by the reorganization of the automata or by interconnecting several simple automata. The use of special automata designed for accepting the so called cyclic strings [12] is also a concern, and will improve further the efficiency of the proposal syntactical method.

References

[1] Robert J. Schalkoff, *Pattern Recognition: statistical, structural and neural approaches*, John Wiley & Sons, Inc., 1992.

[2] R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison-Wesley, Reading, MA, 3rd edition, 1987.

[3] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[4] Gerhard X Ritter and Joseph N Wilson, *Handbook of computer vision algorithms in image algebra*, CRC press LLC, 2nd edition, 2000.

[5] K. S. FU, *Syntactic Methods in Pattern Recognition*, New York: Academic Press, 1974.

[6] Jürgen Schürmann, *Pattern Classification: a unified view of statistical an neural approaches*, John Wiley & Sons, Inc., 1996.

[7] B. D. Ripley, *Pattern Recognition and Neura Networks*, Cambridge University Press, 1996.

[8] João José Neto, "Adaptive rule-driven devices - general formulation and case study," *6 Conference on Implementation and Application of Automata - CIAA 2001*, July 2001.

[9] João José Neto, "Adaptive automata for context-dependet languages," *ACM SIGPLAN Notices*, vol. 29, no. 9, september 1994.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 2nd edition, 2001.

[11] James D. Foley, Andries Van Dam, and Steven K. Feiner, *Introduction to Computer Graphics*, Addison-Wesley, Reading, MA, 1st edition, 1993.

[12] Borivoj MELICHAR, "Deterministic parsing of cyclic strings," *7 Conference on Implementation and Application of Automata - CIAA 2001*, July 2002.

EDUARDO ROCHA COSTA Was Born in São Paulo, Brazil, on March 3, 1970. He received BS degree in Eletrical Engineering from FEI (Faculdade de Engenharia Industrial) São Bernardo, SP, Brazil, in 2001. He is currently a Master student in Electrical Engineering at Escola Politécnica da Universidade de São Paulo. His research interests are in robotics, adaptive automata and their aplications, path planing.

ANDRE RIYUITI HIRAKAWA Was Born in São Paulo, Brazil, on June 15, 1965. He received the BS and MS degrees in Electrical Engineering from São Paulo University, São Paulo, Brazil, in 1990 and 1992, respectively. In 1992, he joined the Department of Computer and Electrical Engineering at the Yokohama National University, Yokohama, Japan, as researcher and also attending the PhD program, concluded in 1997. In 1998, he joined the Department of Computer and Electrical Engineering at the São Paulo University where he is presently an Assistant Professor. His research interests are in Automation and Robotics including Control, intelligent sensor, power electronics, path planning, AGV's, and wireless systems, applied to agricultural and outdoor environment automation.

JOÃO JOSÉ NETO Electical Engineer, Escola Politécnica da Universidade de São Paulo, 1971 M.Sc in Electical Engineering, Escola Politécnica da Universidade de São Paulo, 1975 Ph.D in Electical Engineering, Escola Politécnica da Universidade de São Paulo, 1980 Livre Docente in Electical Engineering, Escola Politécnica da Universidade de São Paulo, 1993 Associate Professor in the Department of Computer Engineering and Digital Systems of the Escola Politécnica da Universidade de São Paulo. Interest Areas: Formal languages, automata, theory of computation, systems programming, compilers, programming languages. Research field: Adaptive devices and their applications.