Adaptive Modeling of Digital Straightness Applied to Geometric Representation Enhancement

Leoncio C. Barros Neto
Engineering School
São Paulo University
São Paulo, Brazil.

André R. Hirakawa
Engineering School
São Paulo University
São Paulo, Brazil.

Antonio M. A. Massola
Engineering School
São Paulo University
São Paulo, Brazil.

ABSTRACT
For representing of digitized straight line segments (DSLS), each of the available research techniques has its advantages and appropriate applications considering the complexities of real world scenarios. Based on adaptive finite automaton (AFA), we propose an alternative paradigm that is convenient for problems modeled by a set of rules. The main objective is to investigate the representation of DSLS through adaptivity, aiming to exploit the ability to represent tolerances, scalability, errors and deviations in angle or in length of the mentioned segments through a device called adaptive DSLS, for short ADSLS. Consequently, ADSLS is shown to be effective to represent segments; furthermore, it is able to adapt, reacting to circumstance stimuli in a single pass.

Keywords
Digital Geometry, Learning and Adaptive Systems, Pattern Recognition, Automata, Classification, Error Recovery.

1. INTRODUCTION
Despite the visible simplicity of digital lines, they are considered fundamental objects in computation [14], in the same way as the concept of straight line is important in Euclidean geometry. Notwithstanding, digitized straight line segments (DSLS[1]) incorporates all the dissimilarities and disparities between the discrete and the continuous representations.

DSLS have different properties from a continuous straight line in Euclidean space. For instance, DSLS cannot be subdivided infinitely to an arbitrarily small segment while maintaining the slope of the original line [18].

In the digitalization process, it is inevitable that continuous straight line segments in Euclidean space be affected by distortion or corruption by noise generating a DSLS in string format with imperfections (not ideal, affected by errors). Fig. 1 shows an example of the string of DSLS in the first quadrant, composed by symbols *a* and *b*.

Accordingly, the traditional DSLS model presents restrictions even nowadays [7] to operate in dynamical scenarios ranging from not accepting changes, in scale, for instance; and there are always interference and noise inducing inaccuracies that are not considered.

Among the available research techniques, statistical, neural and fuzzy logic approaches may be applied to the problem in question.

[1]We use DSLS and other abbreviations, to stand as both the singular and the plural, each one to be grasped from the context.

In particular, [10] comments that most similar is a fuzzy term, emphasizing that when errors inherent to critical scenarios do not follow a known behavior, it is more feasible to use models featuring the best similarity with the ideal model, exemplifying methods based on the theory of fuzzy sets applied to formal languages and automata.

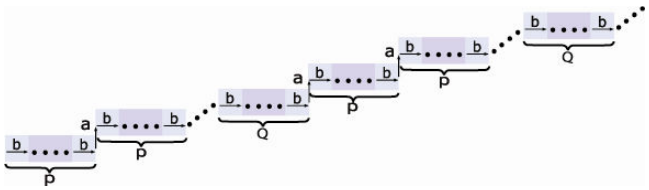

Figure 1 *– A generic SLRD in the first quadrant, composed by runs of P and Q symbols b, as spaced as possible between codes of a, with P and Q constant integers.*

An adaptive method to model similarities in parameters of DSLS aiming to resemble the ideal model is proposed here because, by being able to respond to environmental variable conditions, naturally adaptive devices tend to present the required flexibility to work in dynamic scenarios.

An adaptive device changes its behavior dynamically in response to input stimuli without interferences from other external agents, including users [20]. Normally, they are made by two layers comprising a non-adaptive underlying mechanism ND_0, associated to an adaptive counterpart *AM*, using the same formalism of the first. This growth in complexity profits not only in notable increment in expressive power of the combination, but also in versatility, as one can choose any consolidated mechanism as the non-adaptive device.

It is advantageous to model complex patterns in such a way to benefit from the finite state automaton (FSA) theory background [25]. Owing to this evidence, FSA as ND_0 are used obtaining an adaptive finite automaton (AFA), a Turing-powerful device [22] indicated by Expression 1.

$$\text{AFA}=(ND_0,AM). \quad (1)$$

From Expression 1, the adaptive counterpart *AM* comprises *adaptive actions* responsible for self modification procedures. ND_0 characterizes AFA initial configuration, such as depending on stimulus *i* linked to an operational step *i*, configuration ND_{i-1} is modified by adaptive actions, resulting that the FSA ND_{i-1} is changed into another FSA ND_i belonging to the set of Expression 2.

$$\{ND_0,ND_1,ND_2,ND_3\ldots ND_i\ldots : i \geq 0\}. \quad (2)$$

The reason for changing the machine configurations is to represent, by each configuration, the different model instances,

including the errors involved. This issue of errors in parsing has been studied mainly in relation to compilers, which are within the focus of this study.

This paper is organized as follows. In Section 2 fundaments of AFA are described. Section 3 presents the state-of-the-art, the underlying principles necessary for understanding this study, and codification details, as well. In Section 4, a structural analysis of DSLS is presented. In Section 5, the encoding of adaptive DSLS (ADSLS) is outlined, including the procedure for accessing length similarity. Section 6 exploits facilities provided by ADSLS in some case studies, evaluating the method to conclude in the end. In Section 7, final considerations are drawn.

2. ADAPTIVE FINITE AUTOMATON (AFA)

From Expression 1, AFA is a rule-driven device comprising an adaptive counterpart *AM* consisting of adaptive actions, which are calls to parametric adaptive functions (ADF).
Furthermore, the AFA formalism [20] regards elementary adaptive actions to be applied to the transition set of the automaton, so that sets of elementary adaptive actions are abstracted in ADF which interconnects the adaptive counterpart to ND_i as presented in Fig.2 through generic ADF *R* and *S*.

Fig. 2 shows the static graphic representation of a generic AFA transition where *x* is the current state before the transition; *y* is the current state after the transition; *i* is the input stimulus before the transition; *R* is an ADF executed *before* applying the transition; and, finally, *S* is an ADF executed *after* applying the transition. Graphically, any ADF *R* is portrayed by R• in case it is of the *before* type; likewise, any ADF *S* is an *after* type if it happens to be denoted by •S.

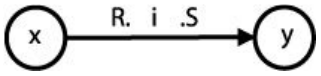

Figure 2 *– A generic AFA transition (x, i) : R→y : S, where R and S are optional.*

When it comes to formats, there are three modalities of elementary adaptive actions indicated in Table 1 by a prefix symbol ?, + or -. In this table, given a certain pattern transition enclosed in brackets, the inspection kind searches the current state set for this pattern; the deletion one erases the pattern from the current state set; and the insertion kind adds the pattern to the current set of transitions. A provision is made that the inspection type is executed first, next the deletion, and finally the insertion kind; adding that null transitions have the lowest priority.

Table 1 – Elementary adaptive action format where R and S are optional and $[(x, i) : R \rightarrow y : S]$ is the pattern to be specified.

Prefix	Meaning	Format
?	Inspection	$?[(x, i) : R \rightarrow y : S]$
-	Deletion	$-[(x, i) : R \rightarrow y : S]$
+	Insertion	$+[(x, i) : R \rightarrow y : S]$

About ADF format, in the general case it has a heading composed by parameters, generators and variables and a body constituted of elementary adaptive actions. All of them are optional; however, if parameters are specified, they have to be supplied to activate the corresponding ADF.

Variables are used in place of any of the components of the elementary adaptive action, further assigned the actual corresponding values in the matching process with the pattern given. Then, after the matching process, variables may be undefined (in the case no match was found) or defined (otherwise). Generators are used to assign names to newly created states. Roughly speaking, they are also like special variables, which are automatically assigned unique values as soon as an ADF is activated. In the activation of an ADF, there occurs the assignment of argument values to the parameters, too. Neither generators nor parameters are allowed to change any longer, once assigned.

To differ from variables, generators receive the symbol * as exponent.

See the format of a hypothetic ADF η, with one generator ger_1, one variable var_1, two parameters α, β and a body of three elementary adaptive actions.

$\eta\ (\alpha, \beta)\ \{ger1^*, var1:$
$?[(r_{i-1}, b) \rightarrow \beta$
$+[(ger1, a) \rightarrow \alpha]$
$-[(var1, \varepsilon) : A \rightarrow r_{i+1}]\}$

η may be activated by a transition such as $(1,a) : \eta\ (2,6) \rightarrow 2$. The adaptive action that activates ADF η happens before the AFA changes its state from state 1 to state 2, as long as a token *a* is received. Concluding, in the same way as ADF η is activated in this example, by choices of attaching sets of ADF to FSA transitions, AFA performance is established, conducting the AFA to accept or reject the input stream.

3. FOUNDANTIONS

The next topic introduces concepts of DSLS and an overview of the state-of-the-art.

3.1 DSLS Background

Chain code was introduced by Freeman in 1970 [9] as a one-pixel-thick boundary descriptor in a grid, and digital straightness was conjectured as well. In this model, given a pixel, the main and immediate neighborhood of this pixel are shown by symbols, as in Fig. 3.

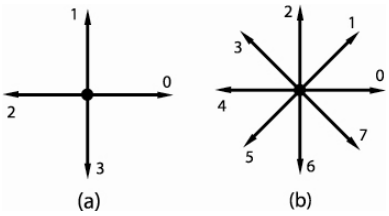

Figure 3 – *On the left is a graphical representation of the chain code symbols 0-3 of neighborhood-4. On the right, of the chain code symbols 0-7 of neighborhood-8*

A digital arc S is understood as a set of interconnected pixels belonging to a digital image, positioned on a grid such that " each point of the set has exactly two neighbors, except two of these points, known as extremes, which have only one neighbor in S" [23].

Hence, restricting ourselves to neighborhood-4 or neighborhood-8, the chain is a sequence of elements where each element is a symbol from Fig. 3 that represents the vector joining two

neighboring pixels of a digital arch, aiming to represent the digital arch in question.

In his model, Freeman stated that strings representing straight lines must obey three properties in neighborhood-8: (*Prop1*) At most two types of symbols, representing directions in the chain code, can be present, and these can differ only by unity, module eight. (*Prop2*) One of the two symbols always occurs singly. (*Prop3*) Successive occurrences of the single symbol are as uniformly spaced as possible among codes of the other value, which occurs in groups.

The meaning of *Prop1* to *Prop3* is to represent the straight line by a sequence of vectors with multiple slope of 45º and the lengths of which are either 1 (when horizontal or vertical), or √2 (when diagonal).

As the third property *Prop3* was considered somewhat unclear, researches proved that the straightness of a digital arc can be determined by the absence of unevenness in its chain code, necessary and sufficient for attending the chord property [12]. The description of the chord property is the following.

Definition 3.1 *Chord Property: A digital arc A is said to have the chord property if for every two digital points c and d in A, and for each point p =(x, y) on* $\overline{cd}$*, there is a point e = (h, k) of A such that max{|x−h|, |y−k|}< 1 where* $\overline{cd}$ *is the line segment between c and d [23].*

Definition 3.1 implied establishing a hierarchical structure composed of consecutive numbers corresponding to the runs and runs of runs of the symbols specified by *Prop1* and *Prop2*. This structure of consecutive numbers is expressed by an additional property *Prop4*: [23] demonstrated that there can be only two possible lengths of these different runs, which are two consecutive integers (for example, P and P+1).

On the other hand, works such as [16] showed examples of DSLS that violate the regularity implicit in the chord property, commenting that, in practice, *Prop3* and *Prop4* are inviable in digital arcs. However, it is more reasonable to expect a slight variation in the runs, within a tolerance level, but always keeping the overall slope, thus delineating an approximate DSLS. Therefore, the criterion used by [16] concentrated in strings that satisfied the first two properties of the conjecture, called *monotonic codes*, as they represent digital arcs that are either ascending or descending, with reference to coordinate axis *x* and *y*.

Estimation of the length of digital segments is another difficulty, there being many length estimators (see [5]). As an introduction to this subject, from [9] the length of a segment codified by string $S = s_1..s_i..s_n$ is given by Expression 3.

$$l_F = (v + h) + s(2)^{\frac{1}{2}} \quad (3)$$

with *v, h* and *s* as representatives of the number of vertical, horizontal and diagonal primitives in *S*, respectively. In reality, the mentioned length can only be approximated, requiring some correction factor ψ to adjust l_F equally to Expression 4.

$$l_E \approx \psi \times l_F \quad (4)$$

where l_E is the estimated length of *S*, after applying the correction factor.

Moreover, three major works guide this proposal. At first, [2] followed an algorithmic procedure similar to that of Freeman which defines discrete lines as digitized Euclidean lines. However, [7] stated that DSLS are very rigid structures, limiting their utilization even after [2] had obtained a certain flexibility. The second, [4] introduced the blurred segments, based on the notions of discrete geometry presented by [21]. Following the arithmetical digital lines presented by [21], [7] proposed an approach to improve the work by [4],"that lost all connection with arithmetic", and by [2].

Adding that the connection of Euclidean geometry with arithmetic discrete geometry takes place in the limit tending to infinity, just as a discrete grid being observed from a point sufficiently far appears to be continuous [8], we are stating an enhanced method by this research taking into account that the adaptive representation can express changes in the scales of segments.

Therefore, an irregular arc may reveal itself as DSLS, provided that it is reviewed in a compatible scale, using metrics. In summary, adaptivity can be an alternative to incorporate the fundamentals of arithmetic discrete geometry to the model of Freeman.

Another key issue in this research is the computational power required to parse DSLS.

3.2 The Syntactic Analysis of DSLS

One of the procedures to determine a syntactic model starts with the definition of a grammar associated with some kind of recognition device [3]. This recognizer is called parser to be account for parsing, deciding if a given observed string belongs to the class represented by the grammar.

However, noise and distortion complicate the computational process of syntactic analysis: apart from distortions, spurious primitives are generated, and real primitives cannot be detected. Moreover, the very natures of the variable periods of symbols codifying DSLS associated with variable lengths are challenges for the syntactic analysis. Context dependencies and changes in orientation angles, with segments of arbitrary length affect the structure of the digital codes of the lines forcing the parser to review its analysis [24].

The understanding of the problem from the syntactic point of view involves the concepts of language, grammar and types of grammars. According to Noam Chomsky, hierarchy dated of 1956, described in [17], languages are classified into four different classes: Recursively Languages (or type 0), Context Sensitive Languages (or Type 1), Context-Free Languages (or type 2) and Regular Languages (or Type 3). There are degrees of complexity related to the classes mentioned since class 3 type is a subset of class Type 2, Type 2 class is a subset of a class type 1, class and type 1 is a subset of Class 0.

Among the existing research approaches, syntactic methods are usually considered unsuitable for tasks involving SLRD. The reason is that SLRD requires powerful context-sensitive grammars, making it impossible to apply simple formalism, such as FSA [6] [14]. Recall that a regular language is specified by a regular grammar. The concepts of regular language and FSA are equivalent in a sense that for every regular language there is at least one FSA that recognizes it and vice versa (see [17] about the formalism of grammars, languages and automata).

The way the languages type 1 and 0 are accepted by the AFA is given by [22]:

> "In the literature, the classical model used for formal acceptance of a language type and 1 and 0 is the Turing machine. Context dependencies existing in these languages can be solved by amendments of its own set of states and transition rules by the AFA"[22].

In principle, the language type 0 would not be part of the scope of this study. However, in order to comply with lengths of DSLS, which can be in various scales, theoretically till infinity, this research implies language type-0.

3.3 Codification

If nothing else is specified, without loss of generality, in this paper neighborhood-4 is the default, so that the symbols of property *Prop1* must be consecutive, module four. More precisely, the symbols that make up strings belong to $\sum=\{a,b,c,d\}$. To satisfy *Prop1*, just consider module 4 along with a=1, b=0, c=3, d=2, for neighborhood-4 of Fig. 3 and Fig. 1.

Any string $S=s_1 \ldots s_n$, $s_i \in \sum$ may be represented by its symbol, followed by the indication of its i-th elemento s_i giving the Expression 5:

$$S: s_i; i = 1,2, \ldots., n. \quad (5)$$

In Expression 5, n denotes the length of string S, which means $|S|=n$. Symbols $s_i \in \sum$ may be called tokens, chain code elements or stimuli, too. The null string $n = 0$ is represented by ε. If all symbols of S are identical, $s=s_1=s_2=\ldots=s_i=\ldots=s_{n-1}=s_n$, a compact representation is $S=s^n$.

The following item presents a brief structural analysis of DSLS through their string formats.

4. STRUCTURAL ANALYSIS

Another method of representing digital lines, applied in this topic, is based on continued fractions, studied by [3]. This mathematical approach provides appropriate models to evaluate errors and approximations in the digitalization process. After the work of [3], continued fractions have been researched, with several developments, described in [14].

Initially, this item reviews the work by [18] evidencing that depending on accuracy requirements, it is possible to derive general formulas for the slope and mathematical models of DSLS. These models are parametric, with parameters calculated in closed form from the slope of DSLS in arbitrary directions on a lattice.

Without loss of generality, we consider chain codes representing a continuous line of orientation angle $\emptyset$ with the positive axis x such that $\emptyset \in [0,\pi/4]$ (indicates that $\emptyset$ belongs to the closed interval between 0 to $\pi/4$ radians) Hence, symbol a occurs alone while symbol b is clustered in the corresponding strings codifying DSLS.

In order to keep the slope of a digital line, the smallest segment of a DSLS is called the Unit of the Straight Line Segment (USLS).

Supposing a DSLS codified by a string S with $|S|=n$. Therefore, S is the concatenation of λ substrings, where $n > \lambda$ and $1 \leq i \leq \lambda$, as each substring is a USLS U given by Expression 6.

$$S: U_i\ ; i = 1,2, \ldots, \lambda. \quad (6)$$

In Expression 6, the null DSLS (λ = 0) is not defined. Besides, we do not account yet that the first and last *USLS* (U_1 and U_λ) may be truncated.

4.1 Models

Taking in to consideration the slope SL as the tangent of an Euclidean segment lying in the first octant, the models are given by the continued fraction of Expression 7,

$$SL = \frac{B}{A} = \frac{1}{P \pm \frac{1}{M \pm \frac{1}{K \pm \cdots}}} \quad (7)$$

where the positive and negative signs are chosen to accelerate the convergence of the fraction and reducing the number of terms (indicative of the order of the model). Additionally, A, B ($B \leq A$), and P are positive integers, while $M,K,\ldots$ are not negative integer.

Two principal models were proposed to describe the pattern arrangement of DSLS symbols. The first was denominated first order model because slope *SL* was a first-order continued fraction. In the first-order model, the line segment traverses B rows and A columns with the slope $SL = B/A = 1/P$; where P, the first-order slope factor, is an integer. It means that A pixels of the line segment are uniformly distributed in B rows with $P = A/B$ pixels in each row.

Hence, each row is a USLS which contains P consecutive pixels. The row-by-row run length representation of the line segment is $PP\cdots P$, or P^B where $P = A/B$.

For the first order model, strings S_U of USLS are given by Expression 8 and shown by Fig. 4.

$$S_U: b^m a; m \geq 1. \quad (8)$$

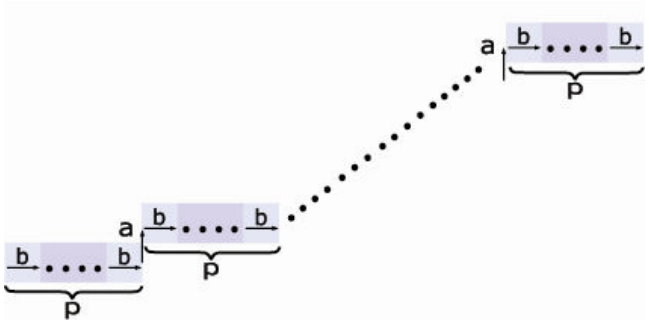

Figure 4 *– First order model, characterized by P as a constant integer.*

In the second model, slope SL is a second order fraction $SL = B \div A = 1 \div [P \pm (1/M)]$; and P is the nearest integer of A / B : P =[A / B]. This means that pixels are adjusted in each USLS by placing P pixels in each of (M - 1) rows, with Q pixels in the remaining row where Q can have only one of following two possibilities: Q = (P+1) or Q = (P – 1). That is, the adjustment is made by decreasing P of a unit, or by increasing P of a unit, as shown by *Prop4*. The row-by-row run-length representation of the line segment in the second order model can be expressed as $P\ldots PQP\ldots PQ$ shown in Fig. 1.

In the second order model, strings S_U of each USLS may be of two kinds:

$$S_U: \begin{cases} b^m a.\ or; \\ b^{m+1}a; m \geq 1\ ou\ b^{m-1}a; m \geq 2 \end{cases}$$

In this second model, the main orientation angle, the one that stands out in the distribution of local angles related to the USLS is θ_S = arctan(1/*P*) with slope 1/*P*. However, USLS occur with inclination 1/*Q* as spaced as possible. Thus, the orientation angle $\emptyset$ with the positive axis *x* of the continuous line that led to the codification will be in the range $\emptyset \in$ [arctan(1/(*P*+1)), arctan(1/*P*)] (assuming *Q* = *P*+1) [13].

4.2 Proposal Summary

This proposal can be summarized in the use of a modified chord property for models of higher orders (order n), thus incorporating tolerances in angle and in length of DSLS. The modified chord property changes neighborhood of Definition 3.1 into a variable neighborhood function such as $\max\{|x-h|,|y-k|\} < n$ where n is the order of the model that depends on the momentary situation and the length of the segment, to sum up, of the stimuli. That is to say, the neighborhood function of DSLS must have a relatively large width, proportional to the measured length towards the overall linear structure [19]. An outline of these higher order models can be seen in Fig. 4 by altering *P* of the first order model to be a variable integer, ranging theoretically till infinity:
$P(a,b^m) : 0 \leq m < \infty$.

5. IMPLEMENTATION OF ADSLS

Regarding techniques for error recovery in this study, in order to exhibit common characteristics to what is being alluded here with other areas, it is often convenient to represent the real numbers in a given circumference and not in a straight line, as usual. Especially, from the circumference of unit length, when defining an arbitrary origin point, one represents any point *T* by its measured distance around the circle in a counterclockwise direction (this by definition). The division of the circle can be from the Farey series in the form of *spyrographs* described on page 326 of [15].
The techniques of error recovery of syntactic analysis of DSLS employ an approach similar to spyrographs in the form of adaptive *loops*, such that, by these loops, the circumference is built by states of the AFA, which moves cyclically and continuously through the closed loop. In effect, adaptive loops have their total number of states according to tolerance levels.

5.1 Starting and Final Structures of DSLS

In order to simplify the description of automata, take in to account that the abbreviation HTST means a sequence head-to-toe of transitions that consumes the same symbol; besides, each state belonging to the sequence may identified by the first state followed by respective sequential index. The extremes of a hypothetic DSLS may be truncated or completely out of the global structural model. In the former case, the sequence should be accepted; in the latter rejected. Fig. 5 exposes an AFA which tests the first USLS of a DSLS $(a^4b)^n$ modeled from Expression 8. Parameter *r4* is the last state of the HTST starting in *r*. From this arrangement, the AFA removes up to four null transitions by ADF RA (see Table 2). Elucidating, each time RA is activated by token *a*, it removes from the automaton one of the null transitions that constitutes the HTST. Furthermore, any token *b* received conducts the AFA to the final state; conditioned to if more than four tokens *a* are received, the sequence is rejected. The analysis of the other extreme is quite similar. Parameter x_i and variables *vr1, vr2, vr3* and *vr4* permit general application of RA in different topologies, as is going to be seen through this paper.

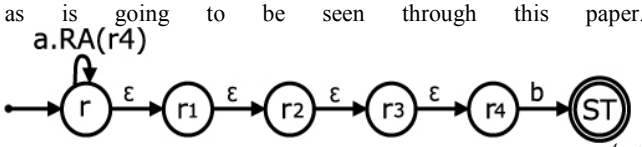

Figure 5 *– AFA for testing the first USLS of a DSLS from* $(a^4b)^n$ *model.*

Table 2 – Parametric ADF RA, of AFA of Fig. 5

$RA(x_i)\{vr1, vr2, vr3, vr4:$
$-[(x_{i-1},\varepsilon) \rightarrow x_i]$
$-[(x_i, vr1) : vr2 \rightarrow vr3 : vr4]$
$+[(x_{i-1}, vr1) : vr2 \rightarrow vr3 : vr4]$
$-[(r,a) \rightarrow a : RA(x_i)]$
$-[(r,a) \rightarrow a : RA(x_{i-1})]$ }

5.2 Slope Errors of DSLS

This topic illustrates the recognition of DSLS subjected to slope errors, exemplifying by $USLS_i = \{a^n b : 3 \leq n \leq 5\}$.
Fig. 6 shows the initial configuration of the automaton prepared to accept truncated $USLS_1$ similar to the last item. ADF are described in Table 3. With the first token *b* consumed, ADF B is activated, which removes transitions of the initial configuration, changing the automaton topology to that of Fig. 7.

Afterwards, the AFA starts to consume the succeeding $USLS_i$: $i > 1$ by each cycle u,u_1,u_2,u_3,u_4,u_5,u in agreement with each $USLS_i$. ADF RB guarantees transitions to states u_4,u_5 will occur after each $USLS_i$ processed, since RA removes transitions to these states.

This process is repeated until the input stream is exhausted. A token *c* is included just to signalize the end of the DSLS, when the automaton reaches the final state if the process is successful. On the other hand, if more than 5 tokens *a* are received, - $[(x_i, vr1) : vr2 \rightarrow vr3 : vr4]$ of RA removes transition of *c* to the final state, rejecting the sequence.

Strings of Fig. 8 and Table 4 show the performance of the AFA. These strings follows the model $USLS_i = \{a^n b: 3 \leq n \leq 5\}$, truncating $USLS_1$ in some strings, too. Strings out of this model are rejected. Note that the AFA performance does not depend on the length of the input DSLS.

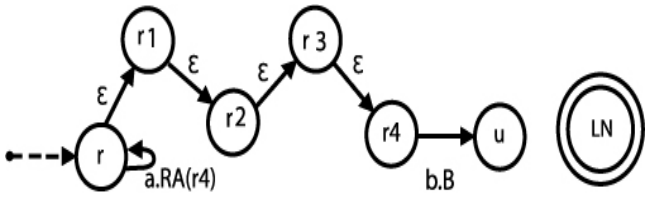

Figure 6 *– Initial Configuration for the automaton to detect a DSLS considering slope errors*

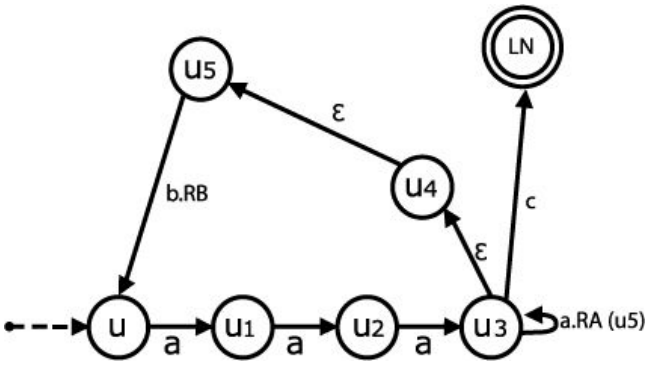

Figure 7 *– Configuration of AFA of Fig. 6 after the activation of ADF B*

Table 3 – ADF of AFA of Fig. 6 and Fig. 7. ADF RA is in table 2

RB{:	B{:
$-[(u_4,\varepsilon)\rightarrow u_5]$	$-[(r,a) \rightarrow r : RA(r_i)]$
$-[(u_3,\varepsilon)\rightarrow u_4]$	$-[(r,\varepsilon) \rightarrow r_1]$
	$-[(r_1,\varepsilon) \rightarrow r_2]$
$+[(u_4,\varepsilon)\rightarrow u_5]$	$-[(r_2,\varepsilon) \rightarrow r_3]$
$+[(u_3,\varepsilon)\rightarrow u_4]$ }	$-[(r_3,\varepsilon) \rightarrow r_4]$
	$+[(u,a) \rightarrow u_1]$
	$+[(u_1,a) \rightarrow u_2]$
	$+[(u_2,a) \rightarrow u_3]$
	$+[(u_3,a) \rightarrow u_3 : RA\ (u_5)]$
	$+[(u_5, \varepsilon) \rightarrow u_4]$
	$+[(u_4, \varepsilon) \rightarrow u_3]$
	$+[(u_5,b) \rightarrow u : RB]$
	$+[(u_3,c) \rightarrow LN]$ }

5.3 DSLS Length Similarity

A method to represent and apply tolerances is by a graph, or loop exemplified in Fig. 9 such that the number of states of the loop (that is, its size) is changed adaptively in function, for example, of angle θ_S related to axis x; besides, θ_S gives the main direction of DSLS S. Fig. 9 shows a loop containing to states, ranging from L_1 to L_{t0}.

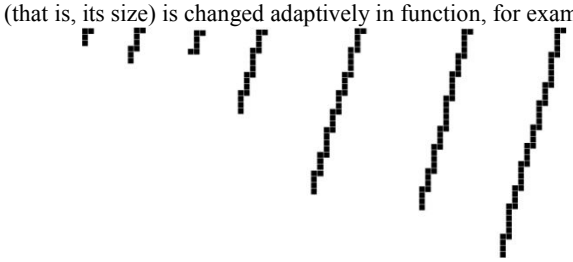

Figure 8 *– Examples of DSLS produced by USLS = $\{a^n b : 3 \leq n \leq 5\}$, accepted by AFA of Fig. 6 and Fig. 7*

Table 4 – Strings of Fig. 8 (from left to right) accepted by AFA of Fig. 6 and Fig. 7.

Strings of Fig. 8	Codification
1	a^2b
2	a^2ba^3b
3	ba^3b
4	$a^3ba^3ba^4ba^3b$
5	$a^3ba^3ba^4ba^3ba^3ba^3ba^4ba^3b$
6	$a^3ba^3ba^4ba^3ba^5ba^4ba^3ba^4b$
7	$a^3ba^5ba^3ba^4ba^3ba^3ba^4ba^3ba^5ba^4b$

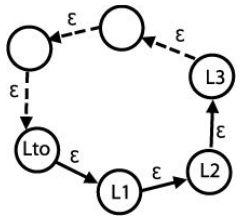

Figure 9 *– A generic cyclic loop.*

Considering that this graph is only for consultation by the automaton, the specific symbol of the transitions between its states becomes irrelevant. The reasoning with the method is that, for each symbol belonging to the DSLS S such that $|S| = n$, the automaton has to access the loop, advancing counterclockwise through the cycle, circulating through the graph as many times as the value of n.

Factor $0 < \psi < 1$ is an error rate meaning a small percentage of n, assuming a simple case in which the length is estimated by the number of symbols in the neighborhood-4, given by Expression 4. Since n is variable depending on lengths, the loop is adapted by changing the amount of states to of Fig. 9, according to Expression 9 and θ_S, the main angle of the DSLS, as well.

$$to \approx [1/(1-\psi)]. \quad (9)$$

For a given value of n, at each turn in the graph, the AFA pumps a primitive thereby obtaining a syntactic measurement parameter $(1-\psi)$ relative to S. It follows that, in the end, n/to symbols would have been pumped with the last symbol of S.

Therefore, $[n/to]$ symbols can be excluded from the total n, to correct the estimated length to l_E. It is also possible to implement inequalities in ADSLS by which to judge lengths of two segments, $|S1| = n$ and $|S2| = m$ in a range of values of Expression 10.

$$n - [n/to] \leq |S2| = m \leq n + [n/to]. \quad (10)$$

6. CASE STUDIES

These case studies center on the classification of basic geometric shapes by ADSLS. Regardless of error causes from the previous stages, i.e. segmentation algorithms that feed an input stream W for classification, we are concerned with tiny errors, abstracted in two broader issues in this item: angle errors of DSLS, and length errors of DSLS. Necessarily, similarity is the essential question to be considered.

Ironically, in shape classification, the most elementary is the segmentation stage in performing attribute-extraction by the selected primitives; not only more complexity is obligatory for the classifier, but also higher computational power, too, and vice versa [26]. Nevertheless, as already seen in item 3.1, modern algorithms usually detect nearly DSLS, while guaranteeing approximate straightness. Hence, the classification step could be concentrated on detecting shape geometric properties affected by existing length errors. In addition, the templates built adaptively by the AFA of these case studies are simpler than the implementation introduced in topic 5, with the advantage of

exhibiting additional flexibilities. They resemble Fig. 10 for the first-quadrant.

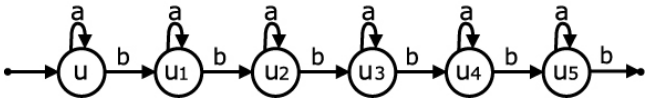

Figure 10 *– A Simplified Model for Automaton Construction. Example with 6 USLS.*

As a consequence, once the segmentation stage accounts for straightness, the AFA for classification would be simpler than the implementations described earlier. Furthermore, it may be convenient, in some systems, to have a simpler classifier at the starting process, without regarding angle errors, to capture some geometric properties quickly by the overall pattern arrangement of USLS, refining the process lately, if necessary.

Next, in the first case study, an automaton that recognizes the square is exemplified, irrespective of length errors.

6.1 ADSLS for Shape Classification

The ADSLS of Fig. 11 and Table 5 recognizes the square. It is an extension of the one in Fig. 6 and 7 utilizing the same ADF RA. An improvement is that each time an $USLS_i$ from the first side of the square is consumed; ADF IB is activated, which constructs the templates of the next 3 sides. These templates are built between states s to t_1: second side, w to w_1: third side, p to p_1: fourth side. In conclusion, from Expression 6, an input stream $W = S_1S_2S_3S_4$, concatenation of four DSLS S_1,S_2,S_3,S_4, can be accepted by the ADSLS in case it codifies the square, such as $\lambda_{S_1} = \lambda_{S_2} = \lambda_{S_3} = \lambda_{S_4}$. Experiments are in the next topic.

6.2 Shape Similarity: Slope Errors of DSLS

To demonstrate the effect of slope errors in classification, the automaton of the last item was fed with strings $W = S_1S_2S_3S_4$ corresponding to Fig. 12-Left, Fig. 12-Right, Fig. 13-Left and Fig.13-Right, varying the slope of individual DSLS.

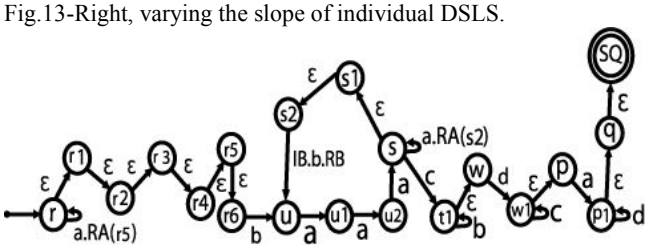

Figure 11 – Initial Configuration of ADSLS that Recognizes the Square without Length Errors

Albeit constituted of DSLS with variable slopes, the string of Fig. 12-Left was accepted by the ADSLS. The first side does not have slope variations; still, combined slope variations from other sides distort the square. Even though the ADSLS is capable of capturing the global geometric property, it accepts the sequence that could be rejected by a more local method. The string of Fig. 12-Right, was accepted. From the previous string, this one introduces slope variations in the first shape side, so that total combination of slope variation from all sides diminishes shape distortions visually. The string of Fig. 13-Left was accepted resulting in a shape with less distortion than those from Fig. 12. In Fig. 13-Left, the first, third and fourth shape sides do not have much slope variations. Slope variations from the third side are difficult to visualize, yet not an ideal DSLS; conversely, the square is visually easily identified. The string of Fig. 13-Right was rejected because USLS (ba^5) is out of the established range. USLS (ba^5) is easily identified visually in the overall formation. Elementary changes in ADF, such as RA, would permit accepting the sequence, adjusting the classifier if required by specifications; otherwise segmentation algorithms should be more precise.

Table 5 – Adaptive Function IB of ADSLS of Fig. 11

IB$\{t_{i+1*}, w_{i+1*}, p_{i+1*}$:
$-[(t_i,\varepsilon)\rightarrow w]$
$-[(w_i,\varepsilon)\rightarrow p]$
$-[(p_i,\varepsilon)\rightarrow q]$
$+[(t_{i+1},b)\rightarrow t_{i+1}]$
$+[(t_i,c)\rightarrow t_{i+1}]$
$+[(t_{i+1},\varepsilon)\rightarrow w]$
$+[(w_{i+1},b)\rightarrow w_{i+1}]$
$+[(w_i,c)\rightarrow w_{i+1}]$
$+[(w_{i+1},\varepsilon)\rightarrow p]$
$+[(p_{i+1},b)\rightarrow p_{i+1}]$
$+[(p_i,c)\rightarrow p_{i+1}]$
$+[(p_{i+1},\varepsilon)\rightarrow q]\}$

6.3 Shape Similarity: Scaling

The ADSLS in Fig. 14 and Table 6, constructed by modifications of previous ADF, recognizes the triangle fed by $W = S_1S_2S_3$. The triangles of Fig. 15 are in different scales, both accepted by the automaton. The accepted triangle may be as big as possible; however, the smallest one is limited to one USLS for each triangle side.

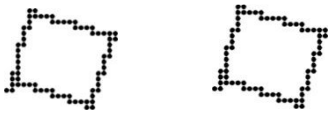

Figure 12 *– Strings $W=S_1S_2S_3S_4$. Left:$[(ba^4)^4b(cb^3)(cb^4)(cb^3)(cb^4)c(dc^3)^4d(ad^3)^3(ad^4)a]$. Right: :$[(ba^3)(ba^4)(ba^3)(ba^4)b(cb^3)(cb^4)(cb^3)(cb^4)c(dc^3)^4d(ad^3)^4a]$.*

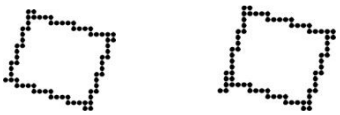

Figure 13 *– Strings $W=S_1S_2S_3S_4$. Left: string $[(ba^3)^4b(cb^3)(cb^4)(cb^3)\ (cb^4)^4c(dc^3)^4d(ad^3)^4a]$. Right: string $[(ba^3)(ba^5)(ba^3)^2b(cb^3)(cb^4)(cb^3)(cb^4)c(dc^3)^4d(ad^3)^4a]$.*

6.4 Shape Similarity: Length error of DSLS

For the triangle, the existence of approximate shapes is informed by $W = \Psi\ S_1S_2S_3$; where S_1 is the pattern string, S_2 and S_3 are observed strings to be compared by Expression 10 with S_1 within the tolerance informed by substring Ψ; essentially Ψ is string $\Psi = y^{t0}$ composed by tokens y, given by the ψ of Expression 9. Any symbol like y not belonging to $\sum$ could be used.

6.4.1 Implementation

Referring to the ADSLS of Fig. 16 showing the loop of Fig. 9 to the right and 3 pointers, p_a, p_c, p_d, it is convenient to clarify that pointers are elementary null transitions; for instance, pointer p_d is pointing to w by the transition $[(p_d,\varepsilon) \rightarrow w]$, called simply by its fixed state p_d. In case tolerance is informed to the ADSLS by tokens y related to Expression 9, ADF RO consumes tokens y, constructing the loop, introducing the 3 pointers, too. Transition

$[(r_1,a) \rightarrow r_1]$ is a simplification to consume possible starting symbols a of $USLS_1$. With first symbol b, the ADSLS begin to consume subsequent $USLS_i$. As already described, each time an $USLS_i$ from the first side of the triangle is consumed, ADF IB and RB are activated. Besides constructing the templates of the next 2 sides between states t_1 to t_2: second side, t_3 to t_4: third side, ADF IB has now the new task to turn the loop by pointer p_a.

Each time *pa* makes a complete turn through the loop, ADF IB executes the following: i) include one new USLS in the loop $[(t_2,b) \rightarrow t_2 : OT]$ and another in $[(w,d) \rightarrow w : OT]$; ii) insert a

Table 6 – Adaptive Functions of ADSLS of Fig. 14

RB{:	IB {t_{3i+1*},t_{3i+2*}, t_{3i+3*},w_{i+1*}:
-[(s_2,ε)→s_1]	-[(t_i,ε)→w]
-[(s_1,ε)→s]	-[(w_{3i},ε)→p]
+[(s_2,ε)→s_1]	+[(t_{3i},c)→ t_{3i+1}]
+[(s_1,ε)→s]}	+[(t_{3i+1},c)→ t_{3i+2}]
	+[(t_{3i+2},b)→ t_{3i+3}]
	+[(t_{3i+2}, c)→ t_{3i+2}]
	+[(t_{3i+3}, ε)→ w]
	+[(w_i , d)→ w_{i+1}]
	+[(w_{i+1}, ε)→ q]}

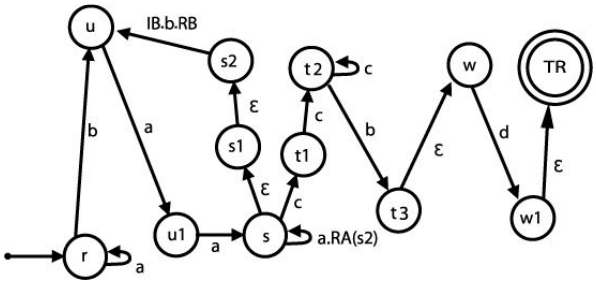

Figure 14 *– Example of ADSLS for Triangle Classification*

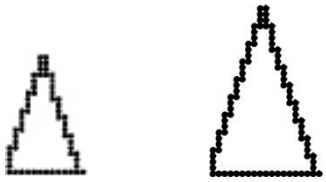

Figure 15 *– Examples of triangles in different scales, classified correctly by ADSLS of Fig. 14. Left: string* $(a^3b)^6(c^3b)(c^4b)(c^3b)^4d^{12}$. *Right: string* $(a^3b)^9(c^3b)(c^4b)(c^3b)^7d^{18}$

transition from the last state of each USLS included before: in the first loop, one that consumes d to t_3; in the second, a null transition to *TR*; iii) moves *pc* to a state just one USLS below in direction to t1, inserting one transition that consumes d from the state pointed by p_c to t_3; iv) moves *pd* to point just one USLS "below" in direction to t_3, inserting one null transition from the state pointed by p_d to *TR*.

Nonetheless, in case ADF OT is activated, it just removes the transition $[(w,\varepsilon) \rightarrow TR]$ to the final state, since the shape is too big, out of tolerance, rejecting the input string. In the same way, too mall shapes are rejected because the ADSLS cannot find either a d transition to t_3, or a null transition to final state *TR*.

6.4.2 Examples

Only two examples of DSLS length errors are provided, since any shape meeting tolerance ψ is accepted, no matter its scale; otherwise it is rejected. Shapes of Fig. 17 were accepted by the ADSLS

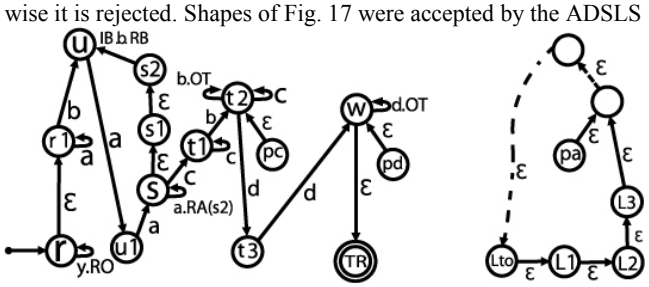

Figure 16 *– Initial configuration of ADSLS for classifications of the triangle affected by length errors*

Figure 17 *– Shapes classified correctly with 20% tolerance. Left:* $(a^3b)^9(c^3b)^8d^{17}$. *Right:* $(a^3b)^9(c^3b)^{11}d^{20}$

with ψ defining a length tolerance of 20%. Both shapes are easily identified visually, though the shape to the right does not close, splitting the contour because of length errors. On the contrary, the one to the left presented a smaller contour than the ideal triangle, leaving the $USLS_1$ of first side out. This contour distortion between the two shapes occurred because, first, the length of the second side changed from 8 x $USLS_i$ to 11 x $USLS_i$; second, the length of third side changed from 17 x d into 20 x d.

6.5 Conclusion

Essentially, angle errors of DSLS have a local character, as compared to length analysis, which is more global, with a greater level of information regarding shapes. In effect, in case (relative) straightness could be guaranteed by the segmentation algorithm, then the classifier should geometrically evaluate the arrangement of $USLS_i$, globally. Thus, in this case, the classifier would be less complex than if it had to perform straightness analyses.

As a consequence of little difficulty to change adaptive actions, the classifier development is relatively simple, as altering the behavior of the classifier to comply with segmentation algorithms needs, too.

Hence, a non-stochastic and flexible application for Syntactic Pattern Recognition based on FSA theory, associated with low sophisticated shape descriptors become an option.

7. FINAL CONSIDERATIONS

To our knowledge, this is the first attempt to introduce AFA in this subject, considering minute errors of DSLS. By traditional techniques, the resulting automaton would have to be implemented *a priori*, with high level of complexity to treat errors that cause imprecise models or imprecise scale of DSLS. These drawbacks have caused the lack of automaton-based solutions in the literature; most results have grammar foundation [3]. An alternative would be to apply fuzzy techniques, such as adaptive fuzzy finite automaton. As reported by [1], models that rely on hidden states are difficult for human experts to understand, increasing complexity.

Furthermore, computational power is another question involved. In the opinion of [11] the capability of a more powerful class of grammars should rejuvenate syntactic research originally pursued in the 70s-80s. Our study showed some capabilities of adaptive techniques related to digital geometry, enabling us to go a step further; emphasizing that adaptivity may indeed contribute to this rejuvenation of the field, as well.

As [24] says, it is considered very difficult to design a one-pass algorithm for identification or generation of DSLS, yet noise-free. As a starting-point, we have outlined a one-pass method by which error tolerances are supplied by previous stages and provided to the AFA. Models of DSLS strings were also presented associated to the corresponding automaton. By such models, a priority for the future is the study of inference algorithms, integrating similarities of frequently repeating DSLS as another parameter of self adaptation and learning by the automaton.

Case studies with basic applications demonstrated the simplicity and efficiency of the method. Compared with other methods of digital line representation, some important advantages of this proposal are that their models are easy to understand, relatively simple to program and flexible to accept changes in their behavior, allowing the use of traditional syntactic tooling. The expressive power of the model incorporates parameters of DSLS such as angle, length and tolerances.

8. ACKNOWLEDGMENTS

Authors thank to Prof. João José Neto who has contributed towards the development of this research.

9. REFERENCES

[1] Gonzalo Bailador and Gracián Triviño. Pattern recognition using temporal fuzzy automata. Fuzzy Sets and Systems, 161(1):37 – 55, 2010. Special section: New Trends on Pattern Recognition with Fuzzy Models.

[2] Partha Bhowmick and Bhargab B. Bhattacharya. Fast polygonal approximation of digital curves using relaxed straightness properties. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29:1590–1602, 2007.

[3] R. Brons. Linguistic methods for the description of a straight line on a grid. Computer Graphics and Image Processing, 3(1):48 – 62, 1974.

[4] I. Debled-Rennesson, F. Feschet, and J. Rouyer-Degli. Optimal blurred segments decomposition of noisy shapes in linear time. Computers & Graphics, 30(1):30 – 36, 2006.

[5] Leo Dorst and Arnold W.M. Smeulders. Discrete straight line segments: Parameters, primitives and properties. Vision Geometry, series Contemporary Mathematics, American Mathematical Society, vol.119:pp.45–62, 1991.

[6] J. Feder. Languages of encoded line patterns. Information and Control, 13(3):230–244, 1968.

[7] F. Feschet. The lattice width and quasi-straightness in digital spaces. In Proceedings of the..., Tampa, FL, 2008. INT. CONFERENCE PATTERN RECOGNITION.

[8] C. Fiorio, D. Jamet, and J. L. Toutant. Discrete circles: an arithmetical approach with non constant thickness. In Electronic Imaging, editor, Proceedings of the..., Volume 6066, San Jose (CA), 2006. SPIE VISION GEOMETRY XIV.

[9] H. Freeman. Boundary encoding and processing. Picture Proceedings and Psychopictorics, pages 241–266, 1970.

[10] J. R. Garitagoitia, J. R. G. de Mendívil, J. Echanobe, J. J. Astrain, and F. Fariña. Deformed fuzzy automata for correcting imperfect strings of fuzzy symbols. IEEE Transactions on Fuzzy Systems, 11(3):299–310, 2003.

[11] Zhu S.-C.b Han, F.a. Bottom-up/top-down image parsing with attribute grammar. IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(1):59–73, 2009.

[12] S.H.Y. Hung. On the straightness of digital arcs. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI- 7(2):203–215, 1985.

[13] Nahum Kiryati and Olaf Kübler. Chain code probabilities and optimal length estimators for digitized three-dimensional curves. Pattern Recognition, 28(3):361–372, 1995.

[14] R. A. Klette and A. B. Rosenfeld. Digital straightness – a review. Discrete Applied Mathematics, 139(1-3):197–230, 2004.

[15] Reinhard Klette and Azriel Rosenfeld. Digital geometry: geometric methods for digital picture analysis. Morgan Kaufmann, 2004.

[16] H. C. Lee and K. S. Fu. Using the FFT to determine digital straight line chain codes. Computer Graphics and Image Processing, 18(4):359–368, 1982.

[17] H.R. Lewis and C.H. Papadimitriou. Elements of the Theory of Computation. Prentice-Hall, 1981.

[18] Shu-Xiang Li and Murray H. Loew. Analysis and modeling of digitized straight-line segments. In Proceedings of the..., pages 294–296, Rome, Italy, 1988. PROCEEDINGS OF INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION.

[19] Peter F.M. Nacken. Metric for line segments. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(12):1312–1318, 1993.

[20] João José Neto. Adaptive rule-driven devices - general formulation and case study. In Springer-Verlag, editor, Proceedings of the..., volume 2494, pages 234–250, Pretoria, South Africa, July 2001. IMPLEMENTATION AND APPLICATION OF AUTOMATA 6TH INTERNATIONAL CONFERENCE, CIAA 2001.

[21] J. P. Reveillès. Géométrie discrète, calcul en nombres entiers et algorithmique. PhD thesis, Université Louis Pasteur, Strasbourg, 1991.

[22] R. L. A. Rocha and J. J. Neto. Autômato adaptativo, limites e complexidade em comparação com máquina de Turing. In Faculdade SENAC de Ciências Exatas e Tecnologia, editor, Proceedings of the..., page 33 a 48. PROCEEDINGS OF THE SECOND CONGRESS OF LOGIC APPLIED TO TECHNOLOGY, 2001.

[23] Azriel Rosenfeld. Digital straight line segments. IEEE Transactions on Computers, C-23(12):1264–1269, 1974.

[24] S. Shlien. Segmentation of digital curves using linguistic techniques. Computer Vision, Graphics and Image Processing, 22(2):277–286, 1983.

[25] N.A. Visnevski, F. Dilkes, S. Haykin, and V. Krishna murthy. Non-self-embedding context-free grammars for multi-function radar modeling-electronic warfare application. In Proceedings of the... IEEE INTERNATIONAL RADAR CONFERENCE, IEEE, 2005.

[26] Kai Ching You and King-Sun Fu. A syntactic approach to shape recognition using attributed grammars. IEEE transactions os systems, man, and cybernetics, 9(6):334–345, June 1979.