# Classification boundary approximation by using combination of training steps for real-time image segmentation.

**J. Mitéran, S. Bouillant, E. Bourennane**
**Le2i** - FRE CNRS 2309 Aile des Sciences de l'ingénieur
Université de Bourgogne
BP 47870
21078 Dijon - FRANCE
miteranj@u-bourgogne.fr

**Abstract**

*We propose a method of real-time implementation of an approximation of the support vector machine decision rule. The method uses an improvement of a supervised classification method based on hyperrectangles, which is useful for real-time image segmentation. We increase the classification and speed performances using a combination of classification methods: a support vector machine is used during a pre-processing step. We recall the principles of the classification methods and we evaluate the hardware implementation cost of each method. We present our learning step combination algorithm and results obtained using Gaussian distributions and an example of image segmentation coming from a part of an industrial inspection problem The results are evaluated regarding hardware cost as well as classification performances. Our contribution can be seen as a real-time implementation of an approximation of the support vector machine and can be generalised to any other classification boundary approximation.*

*Running headline:* SVM approximation for image segmentation

## Introduction

In this paper, we propose a method of approximation of the decision rule of the support vector machine. This approximation allows optimised hardware implementation of the decision boundary, together with an estimation of implementation cost and classification performances. This paper focuses mainly high speed decisions (approximately 10 ns per pixel) which can be useful for image segmentation which can be solved using pixel-wise classification and specific classifiers, for detection of anomalies on manufactured parts, for example. The segmentation is usually the first step of a pattern recognition process.

Classification is a central problem of pattern recognition [1] and many approaches to the problem have been proposed, e.g. neural networks [2], Support Vector Machines (SVM) [3], k-nearest neighbours and kernel-based methods, to name the most common. The chosen classifier must either be implemented in low-cost hardware or in optimised software running in real-time.

It has been proven in the literature that the SVM method gives very good results in many practical cases [4], [5], [6]. However, this robust algorithm is not often used for pixel-wise classification because of the decision rule complexity.

We developed a hyperrectangles-based classifier [7]: this hyperrectangle method belongs to the same family as the NGE algorithm, described by Salzberg [9].

In a previous paper [10], we have shown that it is possible to implement this classifier in a parallel component in order to obtain the required speed, and in another article [11] we indicated that the performances are sufficient for use in a face recognition algorithm. However, the performance of the training step is sometimes affected by ambiguities in the training set, and more generally, the basic hyperrectangles-based method is outperformed by the SVM algorithm.

We propose in this paper an original combination of classifiers allowing for obtaining fast and robust classification applied to image segmentation. The SVM is used during a first step, pre-processing the training set and thus rejecting any ambiguities. The hyperrectangles-based learning algorithm is applied using the SVM classified training set. We will show that the hyperrectangle method imitates the SVM method in terms of performances, for a lower cost of implementation using reconfigurable computing.

In the first part of this paper, we review the principles of the two classifiers: the Hyperrectangles-based method and the SVM. In the second part, we present our combination method and optimisation algorithm. We applied the method on Gaussian distributions, which are often used in literature for performance evaluation of classifiers [12] [1]. Finally, we present practical results obtained in image segmentation of an industrial part.

## Classification algorithms

*Hyperrectangles-based method*

This method divides the attribute space into a set of hyperrectangles for which simple comparators may easily satisfy the membership condition. This hyperrectangle method belongs to the same family as the NGE algorithm, described by Salzberg [9], whose performance was compared to the k-nn method by Wettschereck and Dietterich [8]. The performance of our own implementation was studied in [6].

The training step consists in collecting the set S

$$\mathrm{S}=\left\{\left(\mathbf{x}_1,y_1\right),\left(\mathbf{x}_2,y_2\right),...,\left(\mathbf{x}_p,y_p\right)\right\}$$

of the most representative samples from the various classes and associating a local constraint (hyperrectangle) $H(\mathbf{x}_i)$. Each sample is defined by a feature vector $\mathbf{x}$ in an D dimensional space and its corresponding class $C(\mathbf{x})=y$: $\mathbf{x}=(x_1, x_2, ..., x_D)^T$.

Hyperrectangle determination:

During the first step, an hyperrectangle is build for each sample $\mathbf{x}$ as follows :

Each part $Q_t$ (see Figure 1) defines the area where

$$d_\infty\left(\mathbf{x}_k,\mathbf{x}_l\right)=\left|x_t^k-x_t^l\right| \text{ with}$$

$$d_\infty\left(x,y\right)=\max_{k=1,...,D}\left|x_k-y_k\right|$$

We determine $\mathbf{z}$ as the nearest neighbour belonging to a different class in each part $Q_p$. If $d_p$ is the distance between $\mathbf{x}$ and $\mathbf{z}$ in a given $Q_p$, the limit of the hyperrectangle in the direction is computed as $d_f = d_p.R_p$.

The parameter $R_p$ should be less or equal to 0.5. This constraint ensures that the hyperrectangle cannot contain any sample of opposite classes.

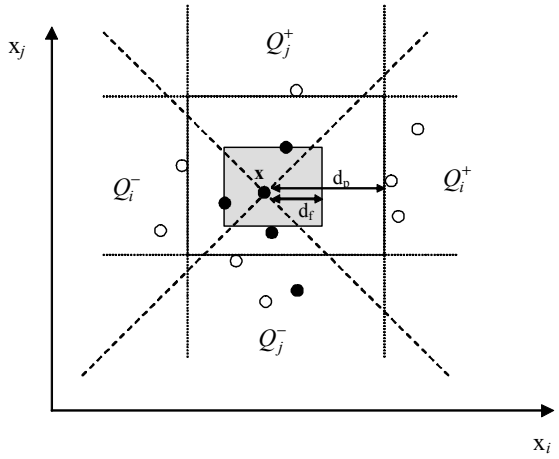



**Figure 1 Hyperrectangle computation**

During the second step, hyperrectangles of a given class are merged together in order to optimise the final number of hyperrectangle [7].

The decision phase consists of allocating a class to a new attribute vector **x**. The membership of a new feature value $x_k$ to a given interval $I_{ik}$ ($i^{th}$ hyperrectangle and $k^{th}$ feature) is easily verified by controlling the two following conditions : $(x_k > a_{ik})$ and $(x_k < b_{ik})$, where $a_{ik}$ and $b_{ik}$ are respectively the lower and upper limits of each polytope or hyperrectangle. Therefore, the verification of the membership of an unknown vector **x** to a class *y* results in a set of comparisons done simultaneously on each feature for every hyperrectangle of class *y*. The resulting decision rule is:

$$C(\mathbf{x}) = y \Leftrightarrow \sum_{i=1}^{i=m_y} \prod_{k=1}^{k=d} \left( (x_k > a_{ik}).(x_k < b_{ik}) \right) \text{ is true (1)}$$

$m_y$ equal to the number of hyperrectangles of class *y* after a merging phase. Sum and product are logical operators. This method is easy to use, and can be implemented for real-time classification using hardware [13] or software optimisation.

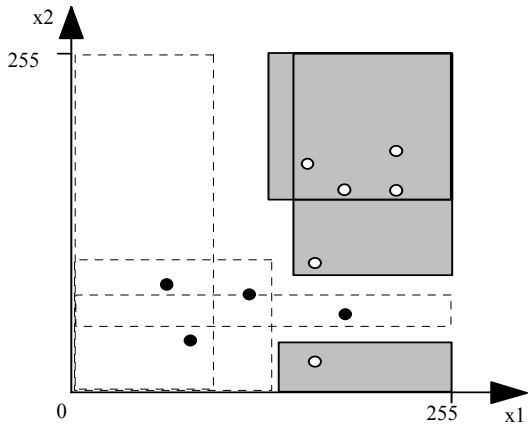



**Figure 2 Hyperrectangles obtained in a two dimensional features space.**

We developed an algorithm allowing evaluation of the implementation cost of this method in Field Programmable Gate Array (FPGA). In recent years FPGAs have become increasingly important and have found their way into system design. FPGAs are used during development, prototyping, and initial production and are replaced by hardwired gate arrays or application specific ICs (ASICs) for highvolume production. This trend is enforced by rapid technological progress, which enables the commercial production of ever more complex devices [14]. The advantage of these components is mainly their reconfigurability [15]. It is possible to integrate the constant values (the limits of hyperrectangles) in the architecture of the decision function. We have coded a tool which automatically generates a VHDL description of a decision function given the result of a training step (i.e. given the hyperrectangles limits). We then have used a standard synthesizer tool for the final implementation in FPGA. We verified that a single comparator between a variable

(feature value) and a constant (hyperrectangle limit) uses only on average a 0.5 slice (using bytes). The slice is the elementary structure of the FPGA of the Virtex family (Figure 3), and one component can contain a few thousand of these blocks. Since the decision rule requires 2 comparators per hyperrectangle and per feature, we evaluate $\lambda_H$, the hardware cost of hyperrectangles implementation (number of slices) with:

$$\lambda_H = d\sum_{y=1}^{y=z} m_y \,, \qquad (2)$$

where z is the number of classes. In the particular case of a 2-class problem, the summation can be computed only to $y$=z-1, since only one set of hyperrectangles defines the boundary.

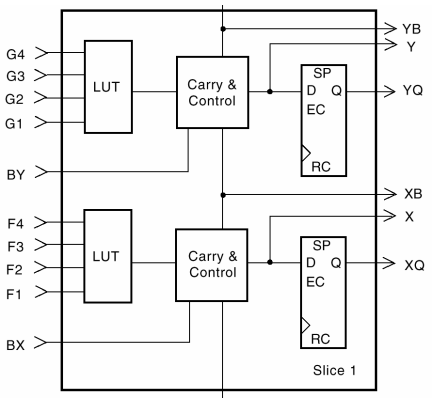



**Figure 3 Slice structure**

We evaluated the performance of this method in various cases, using theoretical distributions [6] as well as real sampling [11]. We compared the performance with neural networks, the Kppv method and a Parzen's kernel based method [1]. It clearly appears that the algorithm performs poorly when the inter-class distances are too small. The overlap between classes is arbitrarily classified thus introducing a classification error.

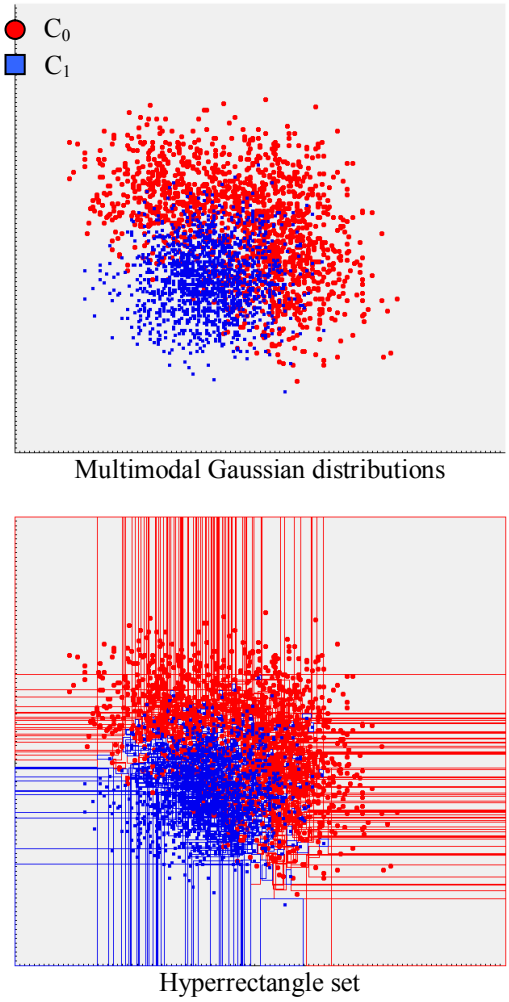



**Figure 4 Gaussian Distributions**

This is shown in Figure 4, where we presented two distributions in a two-dimensional feature space. The C0 class (in red or round dots) is multimodal. The error is the dissymmetric, due to the priority given to the first class. The error rate is 18.88% for the C0 class and 24.34% for the C1 class. Moreover, an important number of hyperrectangles are created in the overlap area, slowing down the decision or increasing the implementation cost.

Many classification methods, such as neural networks, density evaluation-based method and the SVM described above are less sensitive to this overlap. It has been proven in the literature that a particular advantage of SVM over other learning algorithms is that it can be analyzed

theoretically using concepts from computational learning theory. At the same time it can achieve good performance when applied to real problems [16]. We chose this method as a pre-processing step and we will show that it is possible to approximate the result of the SVM using a combination of training steps.

*SVM classification*

A Support Vector Machine (SVM) is a universal learning machine developed by Vladimir Vapnik [3] in 1979. We review here the basic principles, considering here a 2-class problem (whatever the number of classes, it can be reduced, by a "one-against-others" method, to a 2-class problem).

The SVM performs a mapping of the input vectors (objects) from the input space (initial feature space) $R_d$ into a high dimensional feature space $Q$; the mapping is determined by a kernel function $K$. It finds a linear (or non linear) decision rule in the feature space $Q$ in the form of an optimal separating boundary, which is the one that leaves the widest margin between the decision boundary and the input vector mapped into $Q$. This boundary is found by solving the following constrained quadratic programming problem:

Maximize

$$\mathrm{W}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i . \alpha_j . y_i . y_j . K\left(x_\mathrm{i}, x_\mathrm{j}\right) \quad (3)$$

Under the constraints

$$\sum_{i=1}^{n} \alpha_i . y_i = 0$$

and $0 \le \alpha_i \le T$ for i=1, 2, …, n where $x_\mathrm{i} \in R_d$ are the training sample set vectors, and $y_i \in \{-1,+1\}$ the corresponding class label. $T$ is a constant needed for nonseparable classes. $K(u,v)$ is an inner product in the feature space $Q$ which may be defined as a kernel function in the input space. The condition required is that the kernel $K(u,v)$ be a symmetric function which satisfies the following general positive constraint:

$$\iint_{R_d} K(\mathbf{u},\mathbf{v}) g(\mathbf{u}) g(\mathbf{v}) \mathrm{d}\mathbf{u}\, \mathrm{d}\mathbf{v} > 0 \quad (4)$$

which is valid for all $g \neq 0$ for which

$\int g^2(\mathbf{u})\, \mathrm{d}\mathbf{u} < \infty$ (Mercer's theorem).

The choice of the kernel $K(\boldsymbol{u}, \boldsymbol{v})$ determines the structure of the feature space $Q$. A kernel that satisfies (3) may be presented in the form:

$$K(\mathbf{u},\mathbf{v}) = \sum_k a_k \Phi_k(\mathbf{u}) \Phi_k(\mathbf{v}) \quad (5)$$

where $a_k$ are positive scalars and the functions $\Phi_\mathrm{k}$ represent a basis in the space $Q$. Vapnik considered three types of SVMs [3]:

Polynomial SVM:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y} + 1)^p \quad (6)$$

Radial Basis Function SVM:

$$K(\mathbf{x},\mathbf{y}) = e^{\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)} \quad (7)$$

Two-layer neural network SVM:

$$K(\mathbf{x},\mathbf{y}) = Tanh\{k.(\mathbf{x}.\mathbf{y}) - \Theta\} \quad (8)$$

This type of machine generates a kind of two-layer neural network without the back propagation learning step.

The kernel should be chosen a priori. Other parameters of the decision rule (8) are determined by calculating (3), i.e. the set of numerical parameters $\{\alpha_i\}_1^n$ which determines the support vectors and the scalar *b*.

The separating plane is constructed from those input vectors, for which $\alpha_i \neq 0$. These vectors are called *support vectors* and reside on the boundary margin. The number *Ns* of support vectors determines the accuracy and the speed of the SVM. Mapping the separating plane back into the input space $R_d$, gives a separating surface which forms the following nonlinear decision rules:

$$C(\mathbf{x}) = \mathrm{Sgn}\left(\sum_{i=1}^{Ns} y_i \alpha_i \cdot K(\mathbf{s}_i, \mathbf{x}) + b\right) \qquad (9)$$

Where $\mathbf{s}_i$ belongs to the set of *Ns* support vectors defined in the training step.

One can see that the decision rule is easy to compute, but the cost of parallel implementation in ASIC or FPGA is clearly more important than in the case of the hyperrectangles based method. Even if the exponential function can be stored in a particular look up table (LUT) to avoid computation, the scalar product K requires some multiplications and additions; the final decision function requires at least one multiplication and one addition per support vector. For a given model (set of support vectors), it is possible to implement operators using constant values (KCM [17]), as well as we did in the hyperrectangle method. However, the cost of multiplication is significantly more important than the comparator. Chapman[3], [17] proposes a structure using 20 slices per 8 bits multiplier. An 8 bits adder uses 4 slices. The hardware cost of a possible SVM parallel implementation or total number of necessary slices is summarized in table 1. We estimated the number of adders and multipliers needed by a fully parallel computation of K and the final sum of products, in the case of a simplified RBF kernel and a polynomial kernel. Given the number of slices needed by the computation of each elementary operator, we deduced $\lambda_{svm,}$ the hardware cost of each implementation:

**Table 1 SVM hardware cost estimation**

| | | RBF (distance L1) | Polynomial degree p |
|---|---|---|---|
| | | Number of operators | Number of operators |
| K (per support vector) | 16-bit adders (8 slices) | - | d |
| | 8-bit adders (4 slices) | 3d-1 | - |
| | Multiplier Kx8-bit (20 slices) | - | d |
| | Multiplier 8x8-bit (73 slices) | - | p-1 |
| Sum of products | Multiplier Kx16-bit (72 slices) | Ns | Ns |
| | 16 bits adders | 1 | 1 |
| | | Number of slices | Number of slices |
| Total Slices | | $\lambda_{svm} = 72(3d-1)Ns + 8$ | $\lambda_{svm} = (28d + 73(p-1) + 80)Ns + 8$ |

## Combination

*Training method*

Combining decision classifiers is a classical way used in order to increase performances of the general pattern recognition problem [18]. Three main methods are commonly used: sequential, parallel or sequential-parallel combination (Figure 5). These approaches allow

increased performance, but the cost of hardware implementation is high since all the decision functions have to be computed in order to obtain the final classification.

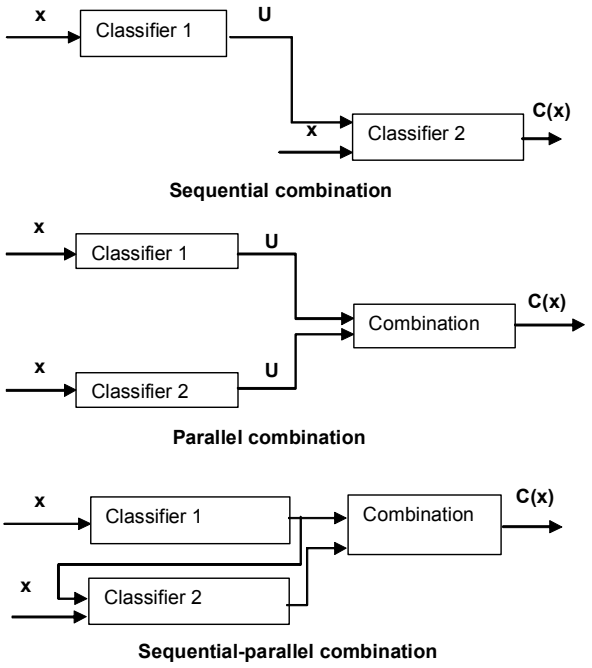



**Figure 5 Combining decision**

More generally, it is possible to combine classification methods during the training step [19]. We propose here such a combination, allowing an approximation of SVM decision boundaries using hyperrectangles (Figure 6).

The SVM method is mainly used here in order to reject ambiguities in the learning set. The algorithm combination is as follows:

- From a training set **S**

$\mathrm{S}=\left\{\left(\mathbf{x}_1,y_1\right),\left(\mathbf{x}_2,y_2\right),...,\left(\mathbf{x}_p,y_p\right)\right\}$, build a model M containing support vectors using SVM algorithm:

$$M=\left\{K,\left(\mathbf{s}_1,y_1\right),\left(\mathbf{s}_2,y_2\right),...,\left(\mathbf{s}_{Ns},y_{Ns}\right),b\right\}$$

- build **S**', the new training set, classifying each sample of S using M and according to eq. 8. :

$$\mathrm{S}'=\left\{\left(\mathbf{x}_1,y'_1\right),\left(\mathbf{x}_2,y'_2\right),...,\left(\mathbf{x}_p,y'_p\right)\right\},$$

- build **H**, set of hyperrectangles using **S**' and the algorithm described in paragraph 0.

During the decision phase, a new test vector **x** is classified regarding **H** and the decision rule (1).

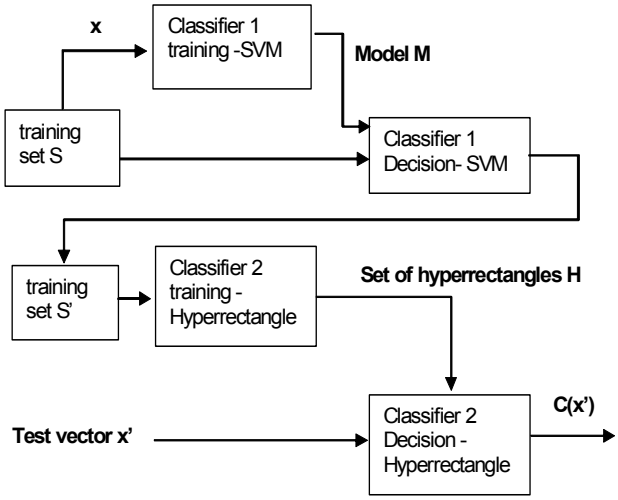
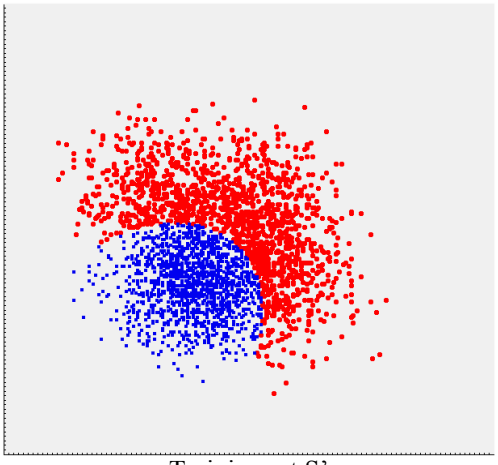


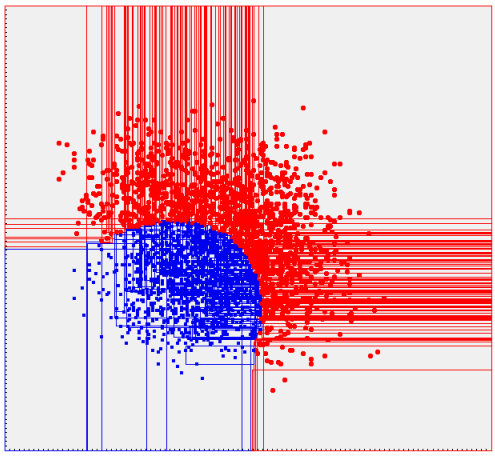

**Figure 6 Combining training steps**

*Application using Gaussian distributions*

We validate the principle of the described method using Gaussian distributions. The tested configuration contains 2 classes. Results are summarised at the end of this part. We used cross-validation with $p$=1000 samples per class for the training set, and $p$=10000 samples per class for the test set. This learning set S is described in the previous paragraph. We use a RBF kernel (eq. 6). The SVM classified set S' and the final set of hyperrectangles are depicted in Figure 7.



Training set S'



Final set of Hyperrectangles

**Figure 7 SVM and Hyperrectangles boundaries**

The results show that the hyperrectangles-based method imitates the SVM decision algorithm, giving a good approximation of boundaries. The error rate of SVM is 16.26% for the C0 class and 13.33% for the C1 class. In this case, the error obtained using final hyperrectangles (learning combination) are 15.60% and 14.20% respectively. One can see that performances are very close, and less dissymmetric than using the initial learning set.

Moreover, the number of hyperrectangles decreased, since the initial numbers were 748 (C0) and 762 (C1) before SVM classification and only 204 (C0) and 201 (C1) after SVM classification. This allows for optimizing the hardware resources in case of implementation. The cost of a direct implementation of SVM decision step is not comparable here, since the number of support vectors is 1190: the estimated hardware cost of SVM is $\lambda_{svm}$= 428 408 slices, whereas the hyperrectangles cost is $\lambda_H$= 205 slices.

An important improvement of performances is obtained, illustrating the good choice of the combination of training steps. However, it is still possible to optimise this result.

*Optimisation*

It is important to note that the final number of hyperrectangles and then the accuracy of SVM boundary approximation depend on *p*, the number of samples of **S**. We defined a method allowing to optimise the implementation minimising $\lambda_H$ . This is done iterating the previous method using randomly chosen subsets of a new training set **S"**. This new training set is obtained adding pseudo-patterns or random vectors to **S**. For each sample **x** of **S'**, we generate a subset **B** of q elements:

$$B = \left\{ \left( \mathbf{x}"_1, y"_1 \right), \left( \mathbf{x}"_2, y"_2 \right), \ldots, \left( \mathbf{x}"_q, y"_q \right) \right\}$$

where $\mathbf{x}_i"=\mathbf{x}_i+\varepsilon$, and $\varepsilon$ is a random value defined in the interval $[x_{ik} -0.1x_{ik}, x_{ik} +0.1x_{ik}]$ for each feature *k*. The class *y"* of each new sample is determined using SVM decision and M Model.

The total number of sample of **S**" is p"=q.p. Increasing the total number of sample improves the accuracy of the boundary approximation: it is clear that a greater number of hyperrectangles will fit more precisely the boundary that a few number of them. In order to find the best compromise between hardware cost and classification

performances, we iterate the method defined in the previous paragraph using subsets of **S**".

The iteration can be stopped either if the classification error stop to decrease of if the maximum of slice ($\lambda_{Hm}$) is reached.

The optimisation algorithm is then

1-from training set **S**, build a model M.

2-build **S**', the new training set, classifying each sample of **S** using M

3-build **S**", oversampling **S** as defined above

4-build **T**, subsampling **S**" using q' sample

5-build **H**, set of hyperrectangles

6-Estimate the classification error *e* using **H** and **S**

7- Estimate $\lambda_H$ and error gradient *eg* from last iteration

8 – Stop if $\lambda_{H>} \lambda_{Hm}$ or if *eg*<*Threshold*, else increase q' and go to 4.

9 – Use the last computed set **H** as the final hyperrectangle set.

*Results*

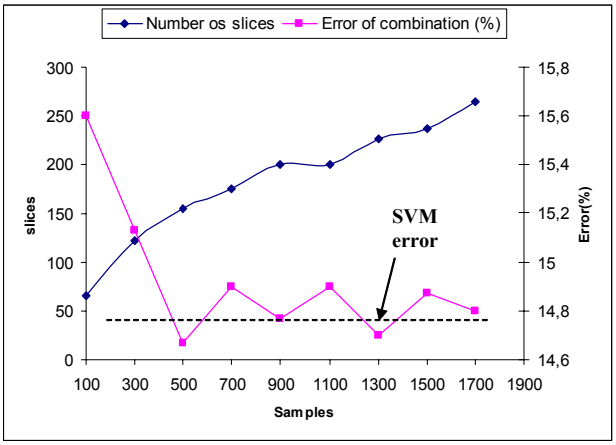
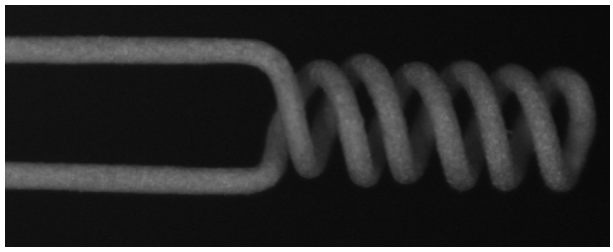



**Figure 8 Results using Gaussian distribution**

The results are summarised in Figure 8. One can see that the classification error of our method converges towards that of the SVM. The optimum number of slices $\lambda_H$ =155 is obtained for q=500.

## Real-time image segmentation for anomalies detection

We applied our method in a preprocessing step of an industrial project of quality control by artificial vision. The parts we have to control are made up of a spiral wire and a non-spiral part called « legs ».



Legs Body (spiraled left)

**Figure 9: Part to be controlled.**

The anomalies existing on the whole part can be grouped into 3 categories:

- Dimensional anomalies: diameter of the wire composing the part, length of the part, length of the body (left spiraled), length of the non-spiraled part.
- Visual anomalies discoloration, stripes, cracks, flaws of surface.

All these anomalies can be found on both the legs and the body of the part.

The third category contains the various anomalies of the spiral not comprising a deterioration of the wire composing the body. We have to distinguish among thirty anomalies at the end of the project. During this preliminary work, we need to obtain a segmented image allowing extraction of high level classification features, such as distance between whorls, whorls surfaces and

orientation etc. Some of these features are depicted in Figure 10, where the part is modeled using whorls orientations and distances.

One can note that the wire is textured: a single threshold could not be a robust operator. We extracted some simple texture features, keeping in mind real-time constraints.

The image size is 1288x1080, and the acquisition rate is 10 images/s.

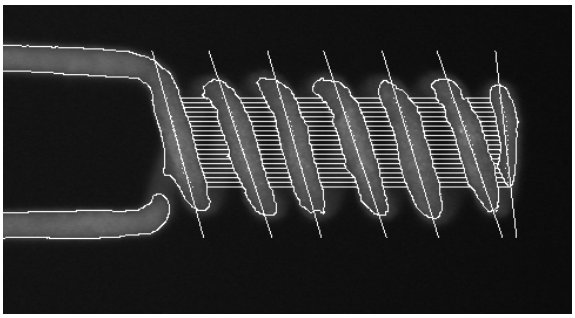


**Figure 10 Part modeling**

A preliminary study of segmentation features led us to choose a four dimensional features space:

$x_0$ is the mean of luminance in 8x8 windows,

$x_1$ is the new value of pixel after local histogram equalisation,

$x_2$ is the mean of a Sobel filter in 8x8 windows,

$x_3$ is the mean of the local contrast in 8x8 windows.

An example of these features is depicted in Figure 12.

The local contrast *V(i,j)* in an [n x m] neighbourhood of pixel *A(i,j)* can be expressed as follows:

$$V(i,j) = \frac{A_{\max} - A_{\min}}{A_{\max} + A_{\min}} \qquad (9)$$

with

$$A_{\max} = \max \left\{ \begin{array}{l} A(i+k, j+l), -\left\lfloor \frac{n-1}{2} \right\rfloor \le k \le \left\lfloor \frac{n}{2} \right\rfloor, \\ -\left\lfloor \frac{m-1}{2} \right\rfloor \le l \le \left\lfloor \frac{m}{2} \right\rfloor \end{array} \right\},$$

$$A_{\min} = \min \left\{ \begin{array}{l} A(i+k, j+l), -\left\lfloor \frac{n-1}{2} \right\rfloor \le k \le \left\lfloor \frac{n}{2} \right\rfloor, \\ -\left\lfloor \frac{m-1}{2} \right\rfloor \le l \le \left\lfloor \frac{m}{2} \right\rfloor \end{array} \right\} \qquad (10)$$

and $\lfloor x \rfloor$ refers to the integer part of *x* (floor operator).

The local mean of the Sobel gradient norm *G(i,j)* and the local mean of luminance *S(i,j)* in a [n x m] neighbourhood of pixels *A(i,j)* can be written as follows:

$$S(i,j) = \frac{1}{mn} \sum_{k=p(n)}^{q(n)} \sum_{l=p(m)}^{q(m)} A(i+k, j+l) \qquad (11)$$

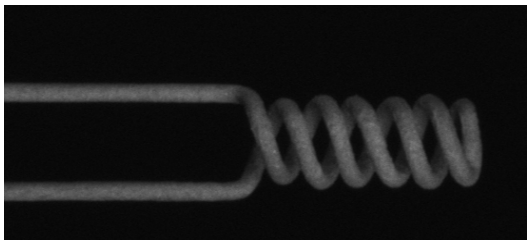
and

$$G(i,j) = \frac{1}{mn} \sum_{k=p(n)}^{q(n)} \sum_{l=p(m)}^{q(m)} g(i+k, j+l) \qquad (12)$$

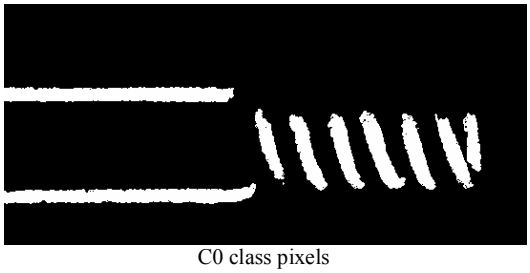
with

$p(n) = -\left\lfloor \frac{n-1}{2} \right\rfloor$, $q(n) = \left\lfloor \frac{n}{2} \right\rfloor$, and *g(i,j)* is the Sobel gradient norm of the pixel *A(i,j)*.

We have chosen this set of features using the SFS [20], [21] algorithm from a superset of 30 features (including variations of windows size and other operators such as morphological operators, local entropy, etc).



Training image



C0 class pixels

C1 class pixels

**Figure 11 Training image and areas**

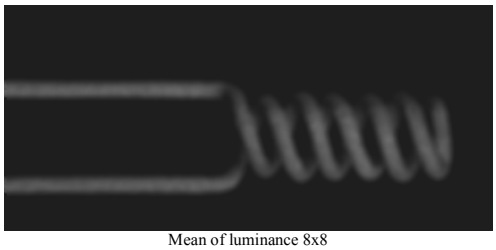
We defined the labels of the training set S manually, using 2 binary images which define respectively the class 0 and class 1 pixels (white pixels in Figure 11). For each class, $p$=5000 pixels or samples were randomly chosen from the white areas of these pictures.

We have depicted two projections of the training set in Figure 13.

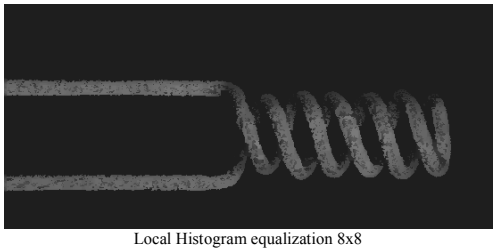
We applied our combination method described in the previous section using 10 test images. An example test image is shown in Figure 15.

The SVM kernel used in this application is RBF.

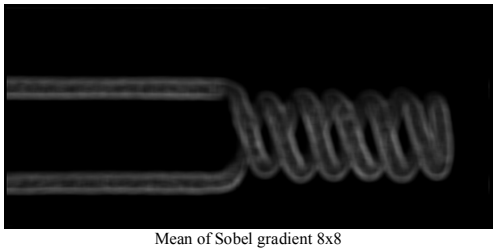
The results of segmentation are depicted in Figure 16. In order to quantify the results, we manually segmented the test images and we computed the classification error of each class for the different segmented images. We obtained the results summarized in Figure 14.



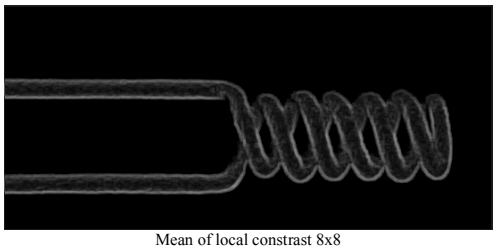
Mean of luminance 8x8



Local Histogram equalization 8x8



Mean of Sobel gradient 8x8



Mean of local constrast 8x8

**Figure 12 Features used for segmentation**

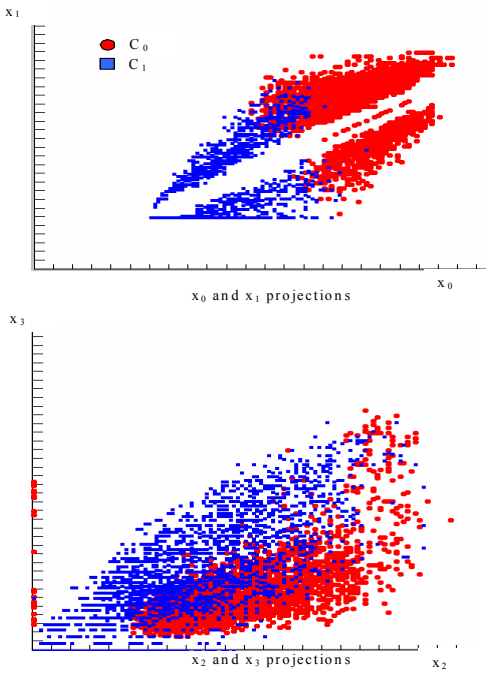



**Figure 13 Training set**

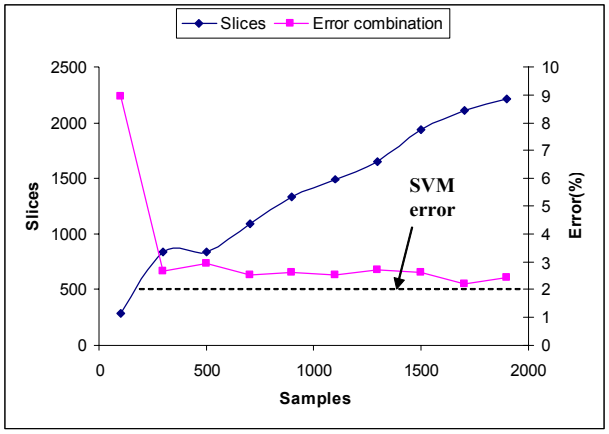



**Figure 14 Performances**

This example illustrates the quality of our combination, since our classification error converges toward the SVM error. The final result of the hyperrectangles-based method is very close to the SVM result (1.94% for SVM and 2.20% for hyperrectangle), and for a lower cost of implementation.

The final implementation needs only $\lambda_H$=2110 slices (one can note that some good results (error is 2.53%) are also obtained with $\lambda_H$=2110 slices). In this particular case, the cost of implementation of SVM is very high, since a total of 475 support vectors were found during the training step. Even in the case of KCM use, the hardware cost of a full parallel decision step is here $\lambda_{svm}$=376 208 !

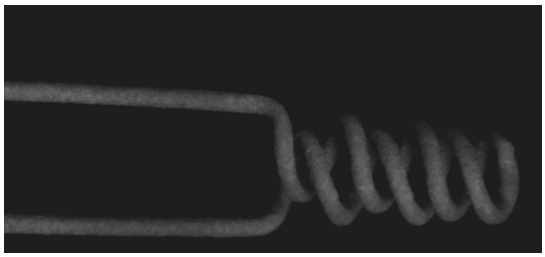


**Figure 15 Test image**

Segmented image using initial learning set and hyperrectangles

Segmented image using SVM

Segmented image using SVM classified learning set and hyperrectangles

**Figure 16 Segmented images**

## Conclusion

We have shown that it is possible to imitate the performance of the SVM classifier for a low cost of implementation combining training steps of SVM and of a particular hyperrectangles-based classifier.

We validated the performance improvement of the basic method in terms of classification as well as in terms of integration cost (or in terms of speed), since the final number of hyperrectangles is minimised.

We demonstrated that it is possible to find an optimum of hardware implementation cost for an error which converges towards the SVM one. We developed the

whole implementation process, from the learning set definition to FPGA implementation using automatic VHDL generation.

One can note that this combination well models our behaviour in front of a problem of quality control by artificial vision: the very first decision given by the expert is often modified for limit cases after observation of the results. This can be seen also as a particular application of ambiguities reject method used in many classification algorithms.

Our future work will be the improvement of the approximation method to other classification boundaries, since this principle can be applied in other case such as complex neural networks.

## References


[1] R. O. Duda and P.E. Hart (1973) *Pattern classification and scene analysis*, Wiley, New York, pp. 230-243.

[2] C. M. Bishop (1995) *Neural networks for Pattern Recognition*, Oxford University Press, pp 110-230.

[3] V. Vapnik (1995) *The nature of statistical learning theory* , Springer-Verlag, New York.

[4] P. Niyogi, C. Burges, P. Ramesh (1999) Distinctive Feature Detection Using Support Vector Machines, ICASSP 99, 1: 425-428.

[5] B. Schölkopf, A. Smola, K.-R. Müller, C. J. C. Burges and V. Vapnik (1998) Support Vector methods in learning and feature extraction, *Australian Journal of Intelligent Information Processing Systems,* 1: 3-9.

[6] K. Jonsson, J. Kittler, Y. P. Li, and J. Matas (1999) Support Vector Machines for Face Authentication. In T. Pridmore and D. Elliman, editors, British Machine Vision Conference, pp 543-553.

[7] J. Mitéran, P. Gorria and M. Robert (1994) Classification géométrique par polytopes de contraintes. Performances et intégration , *Traitement du Signal*, Vol 11 : 393-408.

[8] D. Wettschereck and T. Dietterich (1995) An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms, *Machine Learning,* Vol 19, 1: 5-27

[9] S. Salzberg (1991) A nearest hyperrectangle learning method. *Machine Learning*, 6: 251-276.

[10] M. Robert, P. Gorria, J. Mitéran, S. Turgis (1994) Architectures for real-time classification processor, *Custom Integrated Circuit Conference*, San Diego CA, pp 197-200.

[11] J. Mitéran, J. P. Zimmer, F. Yang, M. Paindavoine (2001) Access control : adaptation and real-time implantation of a face recognition method, *Optical Engineering*, 40(4): 586-593.

[12] B. Dubuisson *Diagnostic et reconnaissance des formes*, HERMES, Paris, 1990.

[13] J. Mitéran, P. Geveaux, R. Bailly and P. Gorria, Real-time defect detection using image segmentation (1997) *Proceedings of IEEE-ISIE 97*, Guimares, Portugal, pp. 713-716.

[14] R. Enzler, T. Jeger, D. Cottet, and G. Tröster (2000) High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs, In *Field-Programmable Logic and Applications* (Proc. FPL

00), Lecture Notes in Computer Science, Vol. 1896, Springer, pp. 525-534

[15] S. Hauck (1998) The Roles of FPGAs in Reprogrammable Systems, Proceedings of the IEEE, 86(4): 615-638.

[16] M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt (1998) Trends and Controversies - Support Vector Machines. *IEEE Intelligent Systems*, 13(4) : 18-28.

[17] K. Chapman (1996) Constant coefficient multipliers for the XC4000E. Xilinx Application, Note XAPP054, Xilinx, Inc.

[18] J. Kittler, M. Hatef, R. P. W. Duin, J. Matas (1998) On combining classifiers in *IEEE transactions on pattern analysis and machine intelligence*, 20(3): 226-239.

[19] B. Moobed (1996) Combinaison de classifieurs, une nouvelle approche, Phd. thesis, Laboratoire d'informatique de polytechnique d'Orsay ; France.

[20] J. Kittler (1978) Feature set search algorithms, *Pattern recognition and signal processing*, Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, pp 41-60.

[21] P. Somol, P. Pudil, J. Novovocova, P. Paclik (1999) Adaptative floating search methods in feature selection, *Pattern Recognition Letters*, 20: 1157-1163.