8

ViSyR: a *Vi*sion *Sy*stem for *R*eal-Time Infrastructure Inspection

Francescomaria Marino[1] and Ettore Stella[2]
[1]*Dipartimento di Elettrotecnica ed Elettronica (DEE) Politecnico di Bari*
[2]*Istituto di Studi sui Sistemi Intelligenti per l'Automazione (ISSIA) CNR*
Italy

1. Introduction

The railway maintenance is a particular application context in which the periodical surface inspection of the rolling plane is required in order to prevent any dangerous situation. Usually, this task is performed by trained personnel that, periodically, walks along the railway network searching for visual anomalies. Actually, this manual inspection is slow, laborious and potentially hazardous, and the results are strictly dependent on the capability of the observer to detect possible anomalies and to recognize critical situations.
With the growing of the high-speed railway traffic, companies over the world are interested to develop automatic inspection systems which are able to detect rail defects, sleepers' anomalies, as well as missing fastening elements. These systems could increase the ability in the detection of defects and reduce the inspection time in order to guarantee more frequently the maintenance of the railway network.
This book chapter presents ViSyR: a patented fully automatic and configurable FPGA-based vision system for real-time infrastructure inspection, able to analyze defects of the rails and to detect the presence/absence of the fastening bolts that fix the rails to the sleepers.
Besides its accuracy, ViSyR achieves impressive performance in terms of inspection velocity. In fact, it is able to perform inspections approximately at velocities of 450 km/h (Jump search) and of 5 km/h (Exhaustive search), with a composite velocity higher than 160 km/h for typical video sequences. Jump and Exhaustive searches are two different modalities of inspection, which are performed in different situations. This computing power has been possible thanks to the implementation onto FPGAs. ViSyR is not only affordable, but even highly flexible and configurable, being based on classifiers that can be easily reconfigured in function of different type of rails.
More in detail, ViSyR's functionality can be described by three blocks: Rail Detection & Tracking Block (RDT&B), Bolts Detection Block (BDB) and Defects Analysis Block (DAB).

- RD&TB is devoted to detect and track the rail head in the acquired video. So doing it strongly reduces the windows to be effectively inspected by the other blocks. It is based on the Principal Component Analysis and the Single Value Decomposition. This technique allows the detection of the coordinates of the center of the rail analyzing a single row of the acquired video sequence (and not a rectangular window having more

rows) in order to keep extremely low the time for I/O. Nevertheless, it allows an accuracy of 98.5%.

- BDB, thanks to the knowledge of the rail geometry, analyses only those windows candidate to contain the fastening elements. It classifies them in the sense of presence/absence of the bolts. This classification is performed combining in a logical AND two classifiers based on different preprocessing. This "cross validated" response avoids (practically-at-all) false positive, and reveals the presence/absence of the fastening bolts with an accuracy of 99.6% in detecting visible bolts and of 95% in detecting missing bolts. The cases of two different kinds of fastening elements (hook bolts and hexagonal bolts) have been implemented.
- DAB focuses its analysis on a particular class of surface defects of the rail: the so-called rail corrugation, that causes an undulated shape into the head of the rail. To detect (and replace) corrugated rails is a main topic in railways maintenance, since in high-speed train, they induce harmful vibrations on wheel and on its components, reducing their lifetime. DAB mainly realizes a texture analysis. In particular, it derives as significant attributes (features) mean and variance of four different Gabor Filter responses, and classifies them using a Support Vector Machine (SVM) getting 100% reliability in detecting corrugated rails, as measured in a very large validation set. The choice of Gabor Filter is derived from a comparative study about several approaches to texture feature extraction (Gabor Filters, Wavelet Transforms and Gabor Wavelet Transforms).

Details on the artificial vision techniques basing the employed algorithms, on the parallel architectures implementing RD&TB and BDB, as well as on the experiments and test performed in order to define and tune the design of ViSyR are presented in this chapter. Several Appendixes are finally enclosed, which shortly recall theoretical issues recalled during the chapter.

2. System Overview

ViSyR acquires images of the rail by means of a DALSA PIRANHA 2 line scan camera [Matrox] having 1024 pixels of resolution (maximum line rate of 67 kLine/s) and using the Cameralink protocol [MachineVision]. Furthermore, it is provided with a PC-CAMLINK frame grabber (Imaging Technology CORECO) [Coreco]. In order to reduce the effects of variable natural lighting conditions, an appropriate illumination setup equipped with six OSRAM 41850 FL light sources has been installed too. In this way the system is robust against changes in the natural illumination. Moreover, in order to synchronize data acquisition, the line scan camera is triggered by the wheel encoder. This trigger sets the resolution along y (main motion direction) at 3 mm, independently from the train velocity; the pixel resolution along the orthogonal direction x is 1 mm. The acquisition system is installed under a diagnostic train during its maintenance route. A top-level logical scheme of ViSyR is represented in Figure 1, while Figure 2 reports the hardware and a screenshot of ViSyR's monitor.

A long video sequence captured by the acquisition system is fed into Prediction Algorithm Block (PAB), which receives a feedback from BDB, as well as the coordinates of the railways geometry by RD&TB. PAB exploits this knowledge for extracting 24x100 pixel windows where the presence of a bolt is expected (some examples are shown in Figure 3).

These windows are provided to the 2-D DWT Preprocessing Block (DWTPB). DWTPB reduces these windows into two sets of 150 coefficients (i.e., $\boldsymbol{D\_LL}_2$ and $\boldsymbol{H\_LL}_2$), resulting

respectively from a Daubechies DWT (DDWT) and a Haar DWT (HDWT). $\boldsymbol{D\_LL_2}$ and $\boldsymbol{H\_LL_2}$ are therefore provided respectively to the Daubechies Classifier (DC) and to the Haar Classifier (HC). The output from DC and HC are combined in a logical AND in order to produce the output of MLPN Classification Block (MLPNCB). MLPNC reveals the presence/absence of bolts and produces a Pass/Alarm signal that is online displayed (see the squares in Figure 2.b), and -in case of alarm, i.e. absence of the bolts- recorded with the position into a log file.

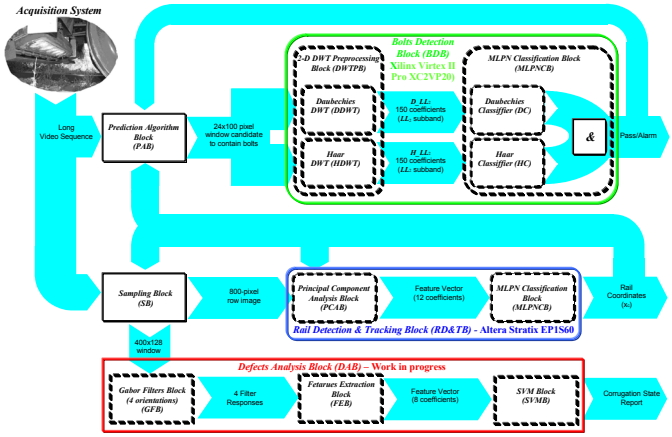


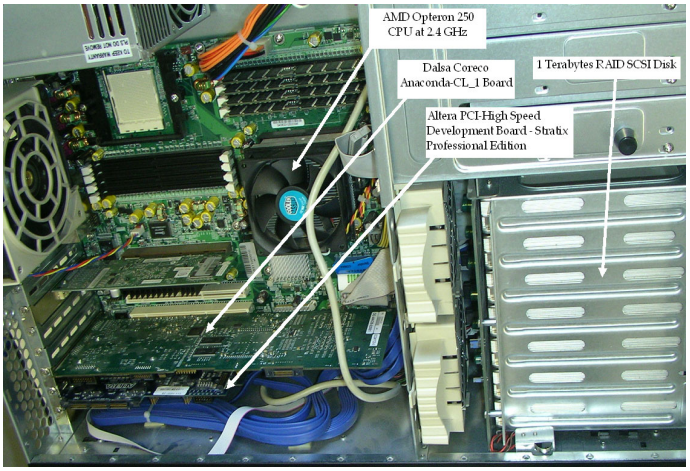
Figure 1. ViSyR's Functional diagram. Rounded blocks are implemented in a FPGA-based hardware, rectangular blocks are currently implemented in a software tool on a general purpose host

RD&TB employs PCA followed by a Multilayer Perceptron Network Classification Block (MLPNCB) for computing the coordinates of the center of the rail. More in detail, a Sampling Block (SB) extracts a row of 800 pixels from the acquired video sequence and provides it to the PCA Block (PCAB). Firstly, a vector of 400 pixels, extracted from the above row and centered on x_c (i.e., the coordinate of the last detected center of the rail head) is multiplied by 12 different eigenvectors. These products generate 12 coefficients, which are fed into MLPNCB, which reveals if the processed segment is centered on the rail head. In that case, the value of x_c is updated with the coordinate of the center of the processed 400-pixels vector and online displayed (see the cross in Figure 2.b). Else, MLPNCB sends a feedback to PCAB, which iterates the process on another 400-pixels vector further extracted from the 800-pixel row.

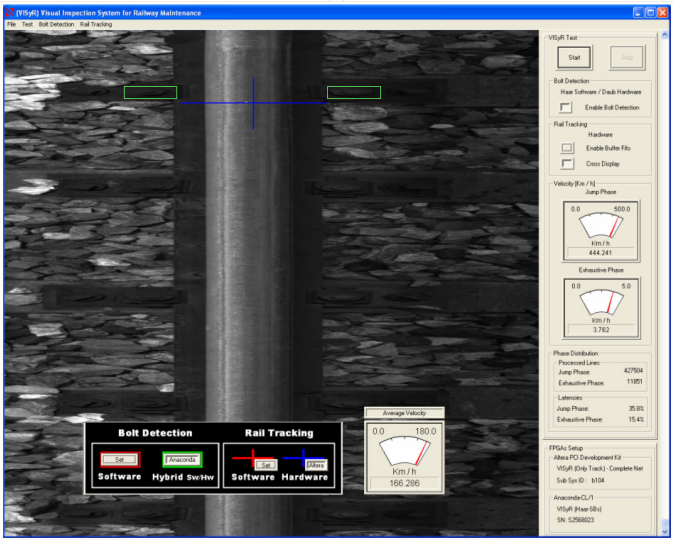
The detected values of x_c are also fed back to various modules of the system, such as SB, which uses them in order to extract from the video sequence some windows of 400x128 pixels centered on the rail to be inspected by the Defect Analysis Block (DAB): DAB convolves these windows by four Gabor filters at four different orientations (Gabor Filters Block). Afterwards, it determines mean and variance of the obtained filter responses and uses them as features input to the SVM Classifier Block which produces the final report about the status of the rail.

BDB and RD&TB are implemented in hardware on an a Xilinx Virtex IITM Pro XC2VP20 (embedded into a Dalsa Coreco Anaconda-CL_1 Board) and on an Altera StratixTM EP1S60 (embedded into an Altera PCI-High Speed Development Board - Stratix Professional

Edition) FPGAs, respectively. SB, PAB and DAB are software tools developed in MS Visual C++ 6.0 on a Work Station equipped with an AMD Opteron 250 CPU at 2.4 GHz and 4 GB RAM.



(a)



(b)

Figure 2. ViSyR: (a) hardware and (b) screenshot

Figure 3. Examples of 24x100 windows extracted from the video sequence containing hexagonal headed bolts. Resolutions along x and y are different because of the acquisition setup

3. Rail Detection & Tracking

RD&TB is a strategic core of ViSyR, since "to detect the coordinates of the rail" is fundamental in order to reduce the areas to be analyzed during the inspection. A rail tracking system should consider that:

- the rail may appear in different forms (UIC 50, UIC 60 and so on);
- the rail illumination might change;
- the defects of the rail surface might modify the rai geometry;
- in presence of switches, the system should correctly follow the principal rail.

In order to satisfy all of the above requirements, we have derived and tested different approaches, respectively based on Correlation, on Gradient based neural network, on Principal Component Analysis (PCA, see Appendix A) with threshold and a PCA with neural network classifier.

Briefly, these methods extract a window ("patch") from the video sequence and decide if it is centred or not on the rail head. In case the "patch" appears as "centred on the rail head", its median coordinate x is assigned to the coordinate of the centre of the rail x_c, otherwise, the processing is iterated on a new patch, which is obtained shifting along x the former "patch".

Even having a high computational cost, *PCA with neural network classifier* outperformed other methods in terms of reliability. It is worth to note that ViSyR's design, based on a FPGA implementation, makes affordable the computational cost required by this approach. Moreover, we have experienced that *PCA with neural network classifier* is the only method able to correctly perform its decision using as "patches" windows constituted by a single row of pixels. This circumstance is remarkable, since it makes the method strongly less dependent than the others from the I/O bandwidth. Consequently, we have embedded into ViSyR a rail tracking algorithm based on PCA with MLPN classifier. This algorithm consists of two steps:

- a data reduction phase based on PCA, in which the intensities are mapped into a reduced suitable space (Component Space);
- a neural network-based supervised classification phase, for detecting the rail in the Component Space.

3.1 Data Reduction Phase.

Due to the setup of ViSyR's acquisition, the linescan TV camera digitises lines of 1024 pixels. In order to detect the centre of the rail head, we discarded the border pixels, considering rows of only 800 pixels. In the set-up employed during our experiments, rail having widths up to 400 pixels have been encompassed.

Matrices **A** and **C** were derived according to equations (A.1) and (A.4) in Appendix A, using 450 examples of vectors. We have selected L=12 for our purposes, after having verified that a component space of 12 eigenvectors and eigenvalues was sufficient to represent the 91% of information content of the input data.

3.2 Classification Phase

The rail detection stage consists of classifying the vector **a**′ -determined as shown in (A.8)- in order to discriminate if it derives from a vector **r**′ centred or not on the rail head. We have implemented this classification step using a Multi Layer Perceptron Neural (MLPN) Network Classifier, since:

- neural network classifiers have a key advantage over geometry-based techniques because they do not require a geometric model for the object representation [A. Jain et al. (2000)];
- contrarily to the id-tree, neural networks have a topology very suitable for hardware implementation.

Inside neural classifiers, we have chosen the MLP, after having experimented that they are more precise than their counterpart RBF in the considered application, and we have adopted a 12:8:1 MLPN constituted by three layers of neurons (input, hidden and output layer), respectively with 12 neurons $n_{1,m}$ (m=0..11) corresponding to the coefficients of **a**′ derived by **r**′ according to (A.7); 8 neurons $n_{2,k}$ (k=0..7):

$$n_{2,k} = f\left(bias_{1,k} + \sum_{m=0}^{11} w_{1,m,k} n_{1,m} \right) \tag{1}$$

and a unique neuron $n_{3,0}$ at the output layer (indicating a measure of confidence on the fact that the analyzed vector **r**′ is centered or not on the rail head):

$$n_{3,0} = f\left(bias_{2,0} + \sum_{k=0}^{7} w_{2,k,0} n_{2,k} \right) \tag{2}$$

In (1) and (2), the adopted activation function $f(x)$, having range]0, 1[, has been:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3}$$

while the weights $w_{1,m,k}$ and $w_{2,k,0}$ have been solved using the Error Back Propagation algorithm with an adaptive learning rate [Bishop. (1995)] and a training set of more than 800 samples (see Paragraph 7.3).

3.3 Rail Detection and Tracking Algorithm

The Rail Detection and Tracking Algorithm consists of determining which extracted vector **r**′ is centred on the rail.

Instead of setting the classifier using a high threshold at the last level and halting the research as soon as a vector is classified as centred on the rail ("rail vector"), we have verified that better precision can be reached using a different approach.

We have chosen a relatively low threshold (=0.7). This threshold classifies as "rail vector" a relatively wide set of vectors **r**′, even when these ones are not effectively centred on the rail (though they contain it). By this way, in this approach, we halt the process not as soon as the first "rail vector" has been detected, but when, after having detected a certain number of contiguous "rail vectors", the classification detects a "no rail". At this point we select as true "rail vector" the median of this contiguous set. In other words, we accept as "rail vector" a relatively wide interval of contiguous vectors, and then select as x_C the median of such interval.

In order to speed-up the search process, we analyse each row of the image, starting from a vector **r**′ centered on the last detected coordinate of the rail centre x_C. This analysis is performed moving on left and on right with respect to this origin and classifying the

vectors, until the begin (x_B) and the end (x_E) of the "rail vectors" interval are detected. The algorithm is proposed in Figure 4.

```
x_C = 512;      // presetting of the coordinate of the centre of the rail
do Start image sequence to End image sequence;
   set r' (400-pixel row) centered on x_C;
   do:
      determine a' (12 coefficients) from r'
      input a' to the classifier and classify r'
      set the new r' shifting 1-pixel-left the previous r'
   while(r' is classified as rail)
// exit from do-while means you have got the begin of the "rail vectors" interval
   x_B = median coordinate of r';
   r' (400-pixel row) centred on x_C;
   do:
      determine a' (12 coefficients) from r'
      input a' to the classifier and classify r'
      set the new r' shifting 1-pixel-right the previous r'
   while(r' is classified as rail)
// exit from do-while means you have got the end of the "rail vectors" interval
   x_E = median coordinate of r';
   output x_C = (x_B+x_E)/2;
end do
```

Figure 4. Algorithm for searching the rail center coordinates

4. Bolts Detection

Usually two kinds of fastening elements are used to secure the rail to the sleepers: hexagonal-headed bolts and hook bolts. They essentially differ by shape: the first one has a regular hexagonal shape having random orientation, the second one has a more complex hook shape that can be found oriented only in one direction.

In this paragraph the case of hexagonal headed bolts is discussed.

It is worth to note that they present more difficulties than those of more complex shapes (e.g., hook bolts) because of the similarity of the hexagonal bolts with the shape of the stones that are on the background. Nevertheless, detection of hook bolts is demanded in Paragraph 7.6.

Even if some works have been performed, which deal with railway problems -such as track profile measurement (e.g., [Alippi *et al.* (2000)]), obstruction detection (e.g., [Sato *et al.* (1998)]), braking control (e.g., [Xishi *et al.* (1992)]), rail defect recognition (e.g., [Cybernetix Group], [Benntec Systemtechnik Gmbh]), ballast reconstruction (e.g., [Cybernetix Group]), switches status detection (e.g., [Rubaai (2003)]), control and activation of signals near stations (e.g., [Yinghua (1994)), etc.- at the best of our knowledge, in literature there are no references on the specific problem of fastening elements recognition. The only found approaches, are commercial vision systems [Cybernetix Group], which consider only fastening elements having regular geometrical shape (like hexagonal bolts) and use geometrical approaches to pattern recognition to resolve the problem. Moreover, these systems are strongly interactive. In fact, in order to reach the best performances, they

require a human operator for tuning any threshold. When a different fastening element is considered, the tuning phase has to be re-executed.
Contrariwise, ViSyR is completely automatic and needs no tuning phase. The human operator has only the task of selecting images of the fastening elements to manage. No assumption about the shape of the fastening elements is required, since the method is suitable for both geometric and generic shapes.

ViSyR's bolts detection is based on MLPNCs and consists of:

- a prediction phase for identifying the image areas (windows) candidate to contain the patterns to be detected;
- a data reduction phase based on DWT;
- a neural network-based supervised classification phase, which reveals the presence/absence of the bolts.

4.1 Prediction Phase

To predict the image areas that eventually may contain the bolts, ViSyR calculates the distance between two adjacent bolts and, basing to this information, predicts the position of the windows in which the presence of the bolt should be expected.
Because of the rail structure (see Figure 5), the distance *Dx* between rail and fastening bolts is constant -with a good approximation- and *a priori* known.
By this way, the RD&TB's task, i.e., the automatic railway detection and tracking is fundamental in determining the position of the bolts along the *x* direction. In the second instance PAB forecasts the position of the bolts along the *y* direction. To reach this goal, it uses two kinds of search:

- Exhaustive search;
- Jump search.

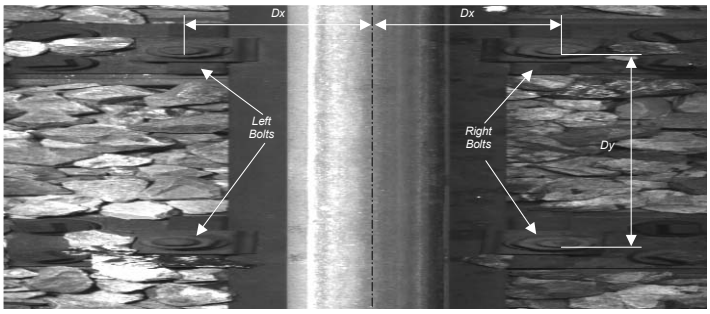


Figure 5. Geometry of a rail. A correct expectation for *Dx* and *Dy* notably reduces the computational load

In the first kind of search, a window exhaustively slides on the areas at a (well-known) distance *Dx* from the rail-head coordinate (as detected by RD&TB) until it finds contemporaneously (at the same *y*) the first occurrence of the left and of the right bolts. At this point, it determines and stores this position (*A*) and continues in this way until it finds the second occurrence of both the bolts (position *B*). Now, it calculates the distance along *y* between *B* and *A* (*Dy*) and the process switches on the Jump search. In fact, the distance along *y* between two adjacent sleepers is constant ad known. Therefore, the Jump search uses *Dy* to jump only in those areas candidate to enclose the windows containing the

hexagonal-headed bolts, saving computational time and speeding-up the performance of the whole system. If, during the Jump search, ViSyR does not find the bolts in the position where it expects them, then it stores the position of fault (this is cause of alarm) in a log-file and restarts the Exhaustive search. A pseudo-code describing how Exhaustive search and Jump search commutate is shown in Figure 6.

```
do Start image sequence to End image sequence;
 repeat
  Exhaustive search;
  if found first left and right bolt store this position (A);
 until found second left and right bolt;
 store this position (B);
 determine the distance along y between B and A;
 repeat
  Jump search
 until the bolts are detected where they were expected;
end do
```

Figure 6. Pseudo code for the Exhaustive search - Jump search commutation

4.2 Data Reduction Phase

For reducing the input space size, ViSyR uses a features extraction algorithm that is able to preserve all the important information about input patterns in a small set of coefficients. This algorithm is based on 2-D DWTs [Daubechies (1988), Mallat (1989), Daubechies (1990 a), Antonini *et al*. (1992)], since DWT concentrates the significant variations of input patterns in a reduced number of coefficients. Specifically, both a compact wavelet introduced by Daubechies [Daubechies (1988)], and the Haar DWT (also known as Haar Transform [G. Strang, & T. Nuguyen (1996)]) are simultaneously used, since we have verified that, for our specific application, the logical AND of these two approaches avoids -almost completely- the false positive detection (see Paragraph 7.5).

In pattern recognition, input images are generally pre-processed in order to extract their intrinsic features. We have found [Stella *et al.* (2002), Mazzeo *et al*. (2004)] that orthonormal bases of compactly supported wavelets introduced by Daubechies [Daubechies (1988)] are an excellent tool for characterizing hexagonal-headed bolts by means of a small number of features[1] containing the most discriminating information, gaining in computational time. As an example, Figure 7 shows how two decomposition levels are applied on an image of a bolt.

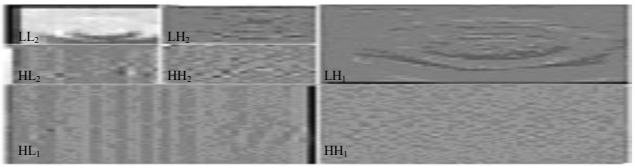


Figure 7. Application of two levels of 2-D DWT on a subimage containing an hexagonal-headed bolt

[1] These are the coefficients of the LL subband of a given decomposition level *l*; *l* depending on the image resolution and equal to 2 in the case of VISyR's set-up.

Due to the setup of ViSyR's acquisition, PAB provides DWTPB with windows of 24x100 pixels to be examined (Figure 3). Different DWTs have been experimented varying the number of decomposition levels, in order to reduce this number without losing in accuracy. The best compromise has been reached by the $\boldsymbol{LL}_2$ subband consisting only of 6x25 coefficients. Therefore, BDB has been devoted to compute the LL2 subbands both of a Haar DWT [G. Strang, & T. Nuguyen (1996)] and of a Daubechies DWT, since we have found that the cross validation of two classifiers (processing respectively $\boldsymbol{D\_LL}_2$ and $\boldsymbol{H\_LL}_2$, i.e., the output of DDWT and HDWT, see Figure 1) practically avoids false positive detection (see Paragraph 7.5). BDB, using the classification strategy described in the following Paragraph, gets an accuracy of 99.9% in recognizing bolts in the primitive windows.

4.3 Classification Phase

ViSyR's BDB employs two MLPNCs (DC and HC in Figure 1), trained respectively for DDWT and HDWT. DC and HC have an identical three-layers topology 150:10:1 (they differ only for the values of the weights). In the following, DC is described; the functionalities of HC can be straightforwardly derived.

The input layer is composed by 150 neurons $D\_n'_m$ (m=0..149) corresponding to the coefficients $\boldsymbol{D\_LL}_2(i, j)$ of the subband $\boldsymbol{D\_LL}_2$ according to:

$$D\_n'_m = \boldsymbol{D\_LL}_2\left(m/25, m \bmod 25\right) \tag{4}$$

The hidden layer of DC consists of 10 neurons $D\_n''_k$ (k=0..9); they derive from the propagation of the first layer according to:

$$D\_n''_k = f\left(D\_bias'_k + \sum_{m=0}^{149} D\_w'_{m,k} D\_n'_m\right) \tag{5}$$

whilst the unique neuron $D\_n'''_0$ at the output layer is given by:

$$D\_n'''_0 = f\left(D\_bias'' + \sum_{k=0}^{9} D\_w''_{k,0} D\_n''_k\right) \tag{6}$$

4where $D\_w'_{m,k}$ and $D\_w''_{k,0}$ are the weights respectively between first/second and second/third layers. The activation function $f(x)$ is the same as (3).

In this scenario, $D\_n'''_0$ ranges from 0 to 1 and indicates a measure of confidence on the presence of the object to detect in the current image window, according to DC.

The outputs from DC and HC ($D\_n'''_0$ and $H\_n'''_0$) are combined as follows:

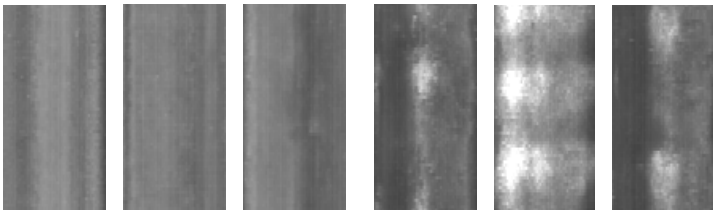
$$\text{Presence} = \left(D\_n'''_0 > 0.9\right) AND \left(H\_n'''_0 > 0.9\right) \tag{7}$$

in order to produce the final output of the Classifier.

The *biases* and the weights were solved using the Error Back Propagation algorithm with an adaptive learning rate [Bishop (1995)] and a training set of more than 1,000 samples (see Paragraph 7.3).

5. Defects Analysis Block

The Defects Analysis Block, at the present, is able to detect a particular class of surface defects on the rail, the so-called rail corrugation. As it is shown in some examples of Figure 8.b, this kind of defect presents a textured surface.



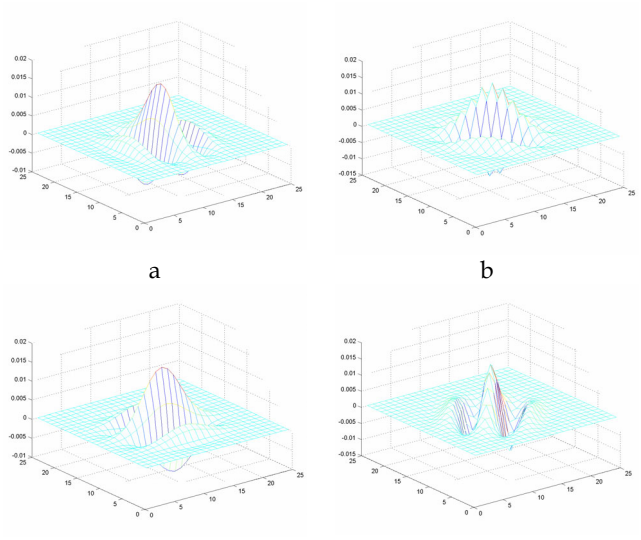
(a) (b)

Figure 8. (a) Examples of rail head; (b) Examples of rail head affected by corrugation

A wide variety of texture analysis methods based on local spatial pattern of intensity have been proposed in literature [Bovik et al. (1990), Daubechies (1990 b)]. Most signal processing approaches submit textured image to a filter bank model followed by some energy measures. In this context, we have tested three filtering approaches to texture feature extraction that in artificial vision community have already provided excellent results [Gong *et al*. (2001), Jain *et al.* (2000)] (Gabor Filters, Wavelet Transform and Gabor Wavelet Transform), and classified the extracted features by means both of a k-nearest neighbor classifier and of a SVM, in order to detect the best combination "feature extractor"/"classifier".
DAB is currently a "work in progress". Further steps could deal with the analysis of other defects (e.g., cracking, welding, shelling ,blob, spot etc.). Study of these defects is already in progress, mainly exploiting the fact that some of them (as cracking, welding, shelling) present a privileged orientation. Final step will be the hardware implementation even of DAB onto FPGA.

5.1 Feature Extraction

For our experiments we have used a training set of 400 rail images of 400x128 pixels centered on the rail-head, containing both "corrugated" and "good" rails, and explored three different approaches, which are theoretically shortly recalled in Appendixes B, C and D.
Gabor Filters. In our applicative context, we have considered only circularly symmetric Gaussians (i.e., $\sigma_x = \sigma_y = \sigma$), adopting a scheme which is similar to the texture segmentation approach suggested in [Jain & Farrokhnia (1990)], approximating the characteristics of certain cells in the visual cortex of some mammals [Porat & Zeevi (1988)].
We have submitted the input image to a filter Gabor bank with orientation 0, $\pi/4$, $\pi/2$ and $3\pi/4$ (see Figure 9), σ=2 and radial discrete frequency F=$\sqrt{2}/2^3$ to each example of the training set. We have discarded other frequencies since they were found too low or too high for discriminating the texture of our applicative context.



a b

c d

Figure 9. Gabor Filters at different orientations: (a) 0; (b) π/4; (c) π/2; (d) 3π/4

The resulting images $i_\theta(x,y)$ (see Figure 10) represent the convolution of the input image $i(x,y)$ with the Gabor filters $h_\theta(x,y)$ where sub index θ indicates the orientation:

$$i_\theta(x,y) = h_\theta(x,y) * i(x,y) \tag{8}$$

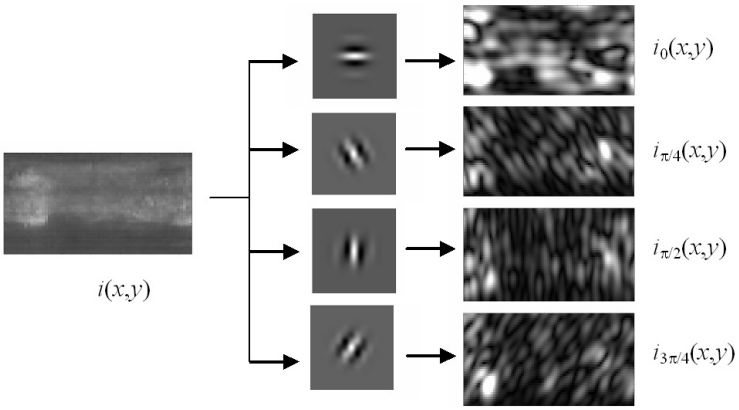


Figure 10. Examples of Gabor Filters ($F=\sqrt{2}/2^3$, σ=2) applied to a corrugated image

Wavelet Transform. We have applied a "Daubechies 1" or "haar" Discrete Wavelet transform to our data set, and we have verified that, for the employed resolution, more than three decomposition levels will have not provided additional discrimination.

Figure 11 shows how three decomposition levels are applied on an image of a corrugated rail.

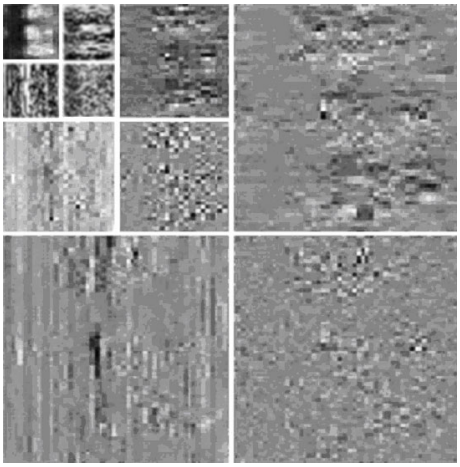


Figure 11. Example of "Daubechies 1" Discrete Wavelet transform (three decomposition levels) of the corrugated image

Gabor Wavelet Transform. A lot of evidence exists for the assumption that representation based on the outputs of families of Gabor filters at multiple spatial locations, play an important role in texture analysis. In [Ma & Manjunath (1995)] is evaluated the texture image annotation by comparison of various wavelet transform representation, including Gabor Wavelet Transform (GWT), and found out that, the last one provides the best match of the first stage of visual processing of humans. Therefore, we have evaluated Gabor Wavelet Transform also because it resumes the intrinsic characteristics both Gabor filters and Wavelet transform.

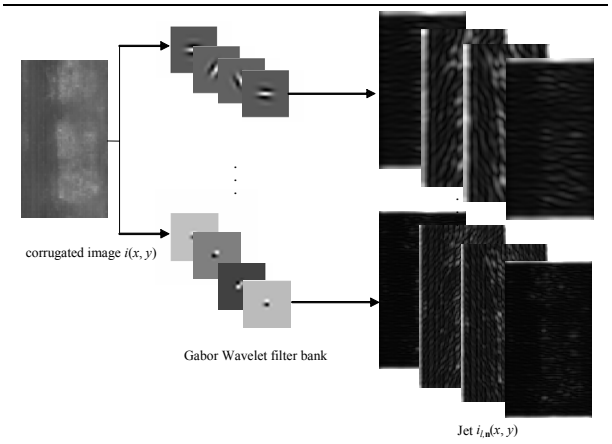


Figure 12. Example of Gabor Wavelet transform of the corrugated image

We have applied the GWT, combining the parameters applied to the Gabor Filter case and to the DWT case, i.e., applying three decomposition levels and four orientations (0, $\pi/2$, $3/4$

π and π, with σ=2 and radial discrete frequency F=$\sqrt{2}/2^3$). Figure 12 shows a set of convolutions of an image affected by corrugation with wavelets based kernels. The set of filtered images obtained for one image is referred to as a "jet".
From each one of the above preprocessing techniques, we have derived 4 (one for each orientation of Gabor filter preprocessing), 9 (one for each subband HH, LH, HL of the three DWT decomposition levels) and 12 pre-processed images $i_p(x,y)$ (combining the 3 scales and 4 orientations of Gabor Wavelet Transform preprocessing). Mean and variance:

$$\mu_p = \iint \left| i_p(x,y) \right| dxdy \tag{9}$$

$$\sigma_p = \sqrt{\iint \left(\left| i_p(x,y) \right| - \mu_p \right)^2 dxdy} \tag{10}$$

of each pre-processed image $i_p(x,y)$ have been therefore used to build the feature vectors to be fed as input to the classification process.

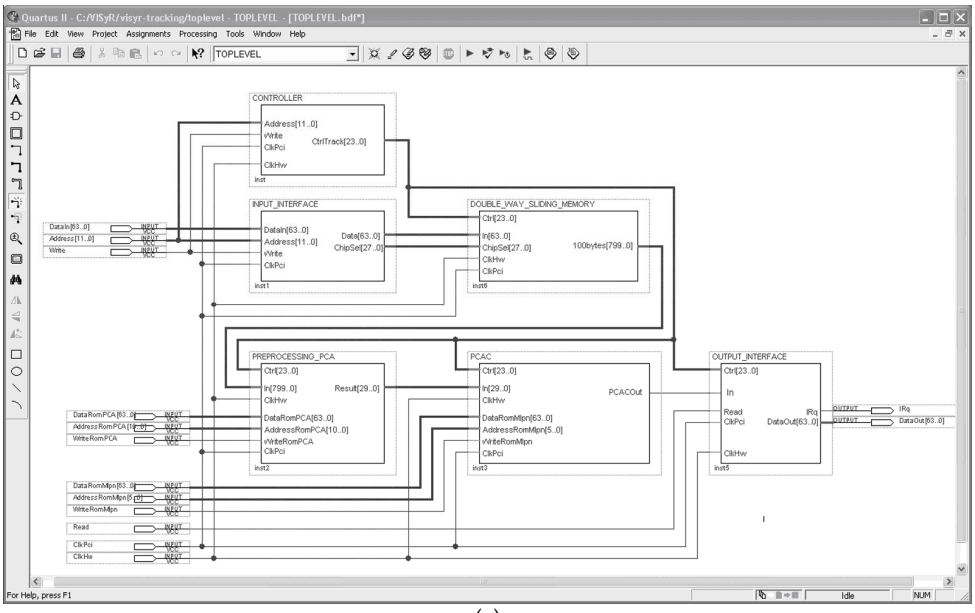
5.2 Classification

We have classified the extracted features using two different classifiers as described in Paragraph 7.8. Considering the results obtained both by k-Nearest Neighbour and Support Vector Machine (see Appendix E), Gabor filters perform better compared to others features extractors. In this context, we have discarded Neural Networks in order to better control the internal dynamic.
Moreover, Gabor filter bank has been found to be preferred even considering the number of feature images extracted to form the feature vector for each filtering approach. In fact, the problem in using Wavelet and Gabor Wavelet texture analysis is that the number of feature images tends to become large. Feature vectors with dimension 8, 18, 24 for Gabor, Wavelet and Gabor Wavelet filters have been used, respectively. In addition, its simplicity, its optimum joint spatial/spatial-frequency localization and its ability to model the frequency and orientation sensitive typical of the HVS, has made the Gabor filter bank an excellent choice for our aim to detect the presence/absence of a particular class of surface defects as corrugation.

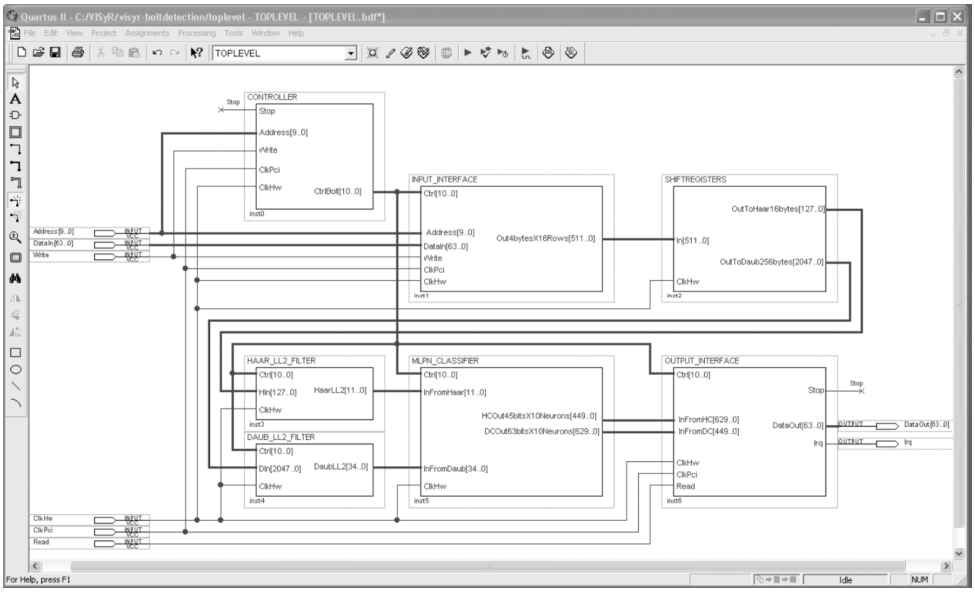
6. FPGA-Based Hardware Implementation

Today, programmable logics play a strategic role in many fields. In fact, in the last two decades, flexibility has been strongly required in order to meet the day-after-day shorter time-to-market. Moreover, FPGAs are generally the first devices to be implemented on the state-of-art silicon technology.
In order to allow ViSyR to get real time performance, we have directly implemented in hardware BDB and RD&TB. In a prototypal version of our system, we had adopted -for implementing and separately testing both the blocks- an Altera's PCI High-Speed Development Kit, Stratix™ Professional Edition embedding a Stratix™ EP1S60 FPGA. Successively, the availability in our Lab of a Dalsa Coreco Anaconda-CL_1 Board embedding a Virtex II™ Pro XC2VP20 has made possible the migration of BDB onto this second FPGA for a simultaneous use of both the blocks in hardware.

A top-level schematic of BDB and RDT&B are provided in Figure 13.a and 13.b respectively, while Figure 14 shows the FPGAs floorplans.



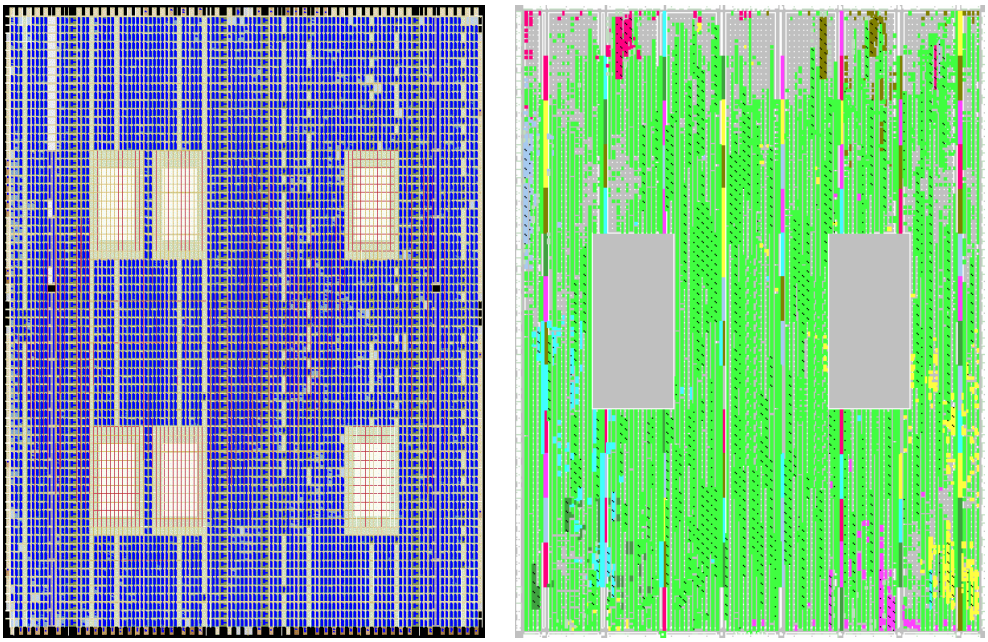
(a)



(b)

Figure 13. A top-level schematic of (a) RD&TB and (b) BDB, as they can be displayed on Altera's QuartusII™ CAD tool

Therefore, even if FPGAs were initially created for developing little glue-logic, they currently often represent the core of various systems in different fields.



(a) (b)

Figure 14. Floorplans of (a) Altera Stratix™ EP1S60 and (b) Xilinx Virtex II™ Pro 20 after being configured

6.1 RD&TB: Modules Functionalities

The architecture can be interpreted as a memory: the task starts when the host "writes" a 800-pixel row to be analyzed. In this phase, the host addresses two shift registers inside the DOUBLE_WAY_SLIDING_MEMORY (pin address[12..0]) and sends the 800 bytes via the input line DataIn[31..0] in form of 200 words of 32 bits.

As soon as the machine has completed his job, the output line irq signals that the results are ready. At this point, the host "reads" them addressing the FIFO memories inside the OUTPUT_INTERFACE.

A more detailed description of the modules is provided in the follow.

Input Interface

The PCI Interface (not explicitly shown in Figure 13.a) sends the input data to the INPUT_INTERFACE block, through DataIn[63..0]. INPUT_INTERFACE separates the input phase from the processing phase, mainly in order to make the processing phase synchronous and independent from delays that might occur during the PCI input. Moreover, it allows of working at a higher frequency (clkHW signal) than the I/O (clkPCI signal).

Double Way Sliding Memory

As soon as the 800 pixel row is received by INPUT_INTERFACE, it is forwarded to the

DOUBLE_WAY_SLIDING_MEMORY, where it is duplicated into 2 shift registers. These shift registers slide in opposite way in order to detect both the end and the begin of the rail interval according to the search algorithm formalized in Figure 4.
For saving hardware resources and computing time, we have discarded the floating point processing mode and we have adopted fixed point precision (see Paragraph 7.7).
By this way, DOUBLE_WAY_SLIDING_MEMORY:

- extracts **r**′ according the policy of Figure 4;
- partitions **r** in four segments of pixels and inputs them to PREPROCESSING_PCA in four trances via 100byte[799..0].

PCA Preprocessing
PREPROCESSING_PCA computes equation (A.7) in four steps. In order to do this, PREPROCESSING_PCA is provided with 100 multipliers, that in 12 clock cycles (ccs) multiply in parallel the 100 pixels (8 bits per pixel) of **r**′ with 100 coefficients of $\mathbf{u}_m$(12 bits per coefficient, m=1..12). These products are combined order to determine the 12 coefficients a_l (having 30 bits because of the growing dynamic) which can be sent to PCAC via Result[29..0] at the rate of 1 coefficient per cc.
This parallelism is the highest achievable with the hardware resources of our FPGAs. Higher performance can be achieved with more performing devices.
Multi Layer Perceptron Neural Classifier
The results of PREPROCESSING_PCA has to be classified according to (1), (2) and (3) by a MLPN classifier (PCAC).
Because of the high hardware cost needed for arithmetically implementing the activation function $f(x)$ -i.e., (3)-, PCAC divides the computation of a neuron into two steps to be performed with different approaches, as represented in Figure 15.

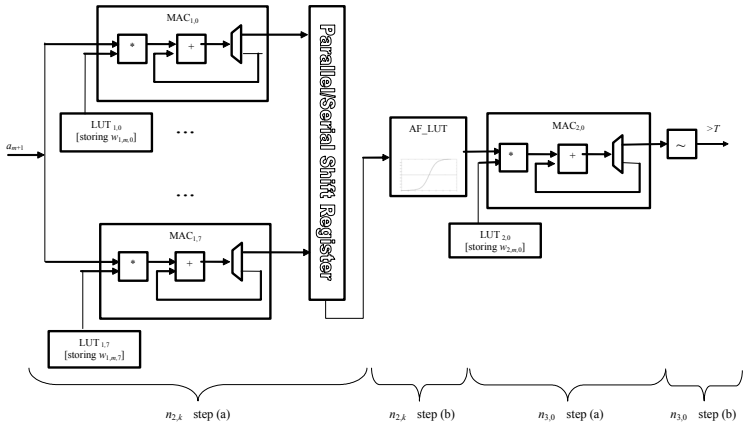


Figure 15. PCAC functionality

Specifically, step (a):

$$x = bias + \sum wn \tag{11}$$

is realized by means of Multiplier-and-ACcumulators (MACs), and step (b):

$$n = f(x) \tag{12}$$

is realized by means of a Look Up Table (for what concerns neurons $n_{2,k}$) and comparers (for what concerns neuron $n_{3,0}$). More in detail:

- neurons $n_{2,k}$, step (a): PCAC has been provided with 8 Multiplier-and-ACcumulators (MACs), i.e., $MAC_{1,k}$ (k=0..7), each one initialized with $bias_k$. As soon as a coefficient a_l (l=1..12) is produced by PREPROCESSING_PCA, the multipliers $MAC_{1,k}$ multiply it in parallel by $w_{1,m,k}$ (m=l+1, k=0..7). These weights have been preloaded in 8 LUTs during the setup, $LUT_{1,k}$ being related to $MAC_{1,k}$ and storing 12 weights. The accumulation takes 12 ccs, one cc for each coefficient a_l coming from PREPROCESSING_PCA; at the end of the computation, any $MAC_{1,k}$ will contain the value x_k.
- neurons $n_{2,k}$, step (b): The values x_k are provided as addresses to AF_LUT through a parallel input/serial output shift register. AF_LUT is a Look up Table which maps at any address x the value of the Activation Function $f(x)$. The adopted precision and sampling rate are discussed in Paragraph 7.4.
- neuron $n_{3,0}$, step (a): This step is similar to that of the previous layer, but it is performed using a unique $MAC_{2,0}$ which multiplies $n_{2,k}$ (k=0..7) by the corresponding $w_{2,k,0}$ at the rate of 1 data/cc.
- neuron $n_{3,0}$, step (b): Since our attention is captured not by the effective value of $n_{3,0}$, but by the circumstance that this might be greater than a given threshold T=0.7 (the result of this comparison constitutes the response of the classification process), we implement step (b) simply by comparing the value accumulated by $MAC_{2,0}$ with $f^{-1}(T)$.

Output Interface

Because of its latency, PCAC classifies each pattern 5 ccs after the last coefficient is provided by PREPROCESSING_PCA. At this point, the single bit output from the comparer is sent to OUTPUT_INTERFACE via PCACOut.

This bit is used as a stop signal for two counters. Specifically, as soon as a value "1" is gotten on PCACOut, a first counter C_B is halted and its value is used for determining which position of the shift of the DOUBLE_WAY_SLIDING_MEMORY is that one centered at the begin of the "rail vector" interval. Afterward, as soon as a value "0" is received from PCACOut, a second counter C_E is halted signaling the end of the "rail vector" interval. At this point, Irq signals that the results are ready, and the values of C_B and C_E packed in a 64 bits word are sent on DataOut[63..0]. Finally, the host can require and receive these results (signal read).

6.2 BDB: Modules Functionalities

Similarly to RD&TB, even BDB can be interpreted as a memory which starts its job when the host "writes" a 24x100 pixel window to be analysed. In this phase, the host addresses the dual port memories inside the INPUT_INTERFACE[2] (pins address[9..0]) and sends the 2400 bytes via the input line data[63..0] in form of 300 words of 64 bits. As soon as the machine has completed his job, the output line irq signals that the results are ready. At this point, the host "reads" them addressing the FIFO memories inside the OUTPUT_INTERFACE.

[2] In addition, INPUT_INTERFACE aims at the same goals of decoupling the input phase from the processing phase, as previously said in the case of RD&TB.

Daubechies DWT Preprocessing

Daubechies 2-D DWT preprocessing is performed by the cooperation of the SHIFTREGISTERS block with the DAUB_LL2_FILTER block.

Even in this case, we have discarded the floating point processing mode and we have adopted fixed point precision (see Paragraph 7.7). Moreover, since we are interested exclusively on the LL_2 subband, we have focused our attention only on that.

It can be shown that, for the 2-D DWT proposed by Daubechies in [Daubechies (1988)] having the 1-D *L* filter:

0,035226	-0,08544	-0,13501	0,45988	0,80689	0,33267

(13)

the LL_2 subband can be computed in only one bi-dimensional filtering step (instead of the classical twice-iterated two monodimensional steps shown in Figure 23 in Appendix C), followed by a decimation by 4 along both rows and columns. Figure 16 reports the applied symmetrical 16x16 kernel.

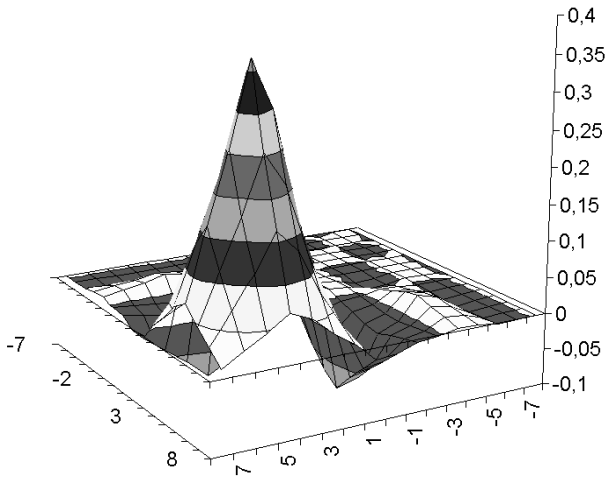


Figure 16. Symmetrical 16x16 kernel for directly computing in one 2-D step the LL_2 subband of the DWT based on the 1-D low-pass filter . The filtering has to be followed by decimation by 4 along both rows and columns

We decided of computing LL_2 directly in only one 2-D step, because:

- this requires a controller much simpler than the one used by the separable approach (Figure 23, in Appendix C);
- separable approach is greatly efficient in computing all the four subbands of each level. But ViSyR's classification process does not need other subbands than LL_2;
- when fixed point precision is employed, each step of the separable approach produces results with different dynamic, so doing, the hardware used at a certain step becomes unusable for implementing the further steps;
- the error (due to the fixed point precision) generated in a unique step does not propagate itself and can be easily controlled. Conversely, propagation occurs along four different steps when LL_2 is computed by means of separable approach.

In this scenario, SHIFTREGISTERS implements a 16x16 array which slides on the 24x100 input window shifting by 4 along columns at any clock cycle (cc). This shift along columns is realized by a routing among the cells as that one shown in Figure 17, that represents the j^{th} row (j=0..15) of SHIFTREGISTERS.

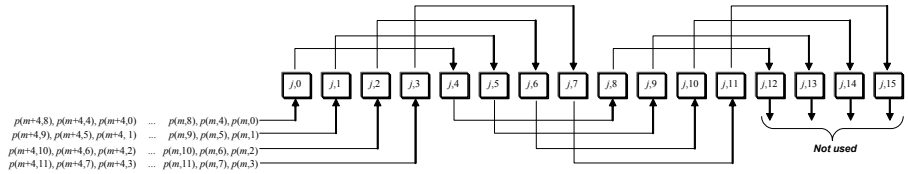


Figure 17. The j^{th} row of the array of 16x16 shift registers in the SHIFTREGISTERS block. Each square represents an 8-bit register

The shift by 4 along the rows is performed by INPUT_INTERFACE which feeds into the j^{th} row of the array only the pixels $p(m, n)$ of the 24x100 input window (m=0..23, n=0..99) where:

$$j \bmod 4 = m \bmod 4 \tag{14}$$

At any cc, sixteen contiguous rows of the input window are fed in parallel into SHIFTREGISTERS at the rate of 64 bytes/cc (4 bytes of each row for 16 rows) through IN[511..0]. Simultaneously, all the 256 bytes latched in the 16x16 array are inputted in parallel into DAUB_LL2_FILTER through OutToDaubLL256bytes[2047..0]. DAUB_LL2_FILTER exploits the symmetry of the kernel (see Figure 16), adding the pixels coming from the cells (j, l) to those ones coming from the cells (l, j) (j=0..15, l=0..15); afterwards, it computes the products of these sums and of the diagonal elements of the array by the related filter coefficients, and, finally, it accumulates these products.
As a result, DAUB_LL2_FILTER produces the $\boldsymbol{LL}_2$ coefficients after a latency of 11 ccs and at the rate of 1 coefficient/cc. These ones are now expressed in 35 bits, because of the growing of the dynamic, and are input into MLPN_CLASSIFIER via InFromDaub[34..0].
We are not interested in higher throughput, since -because of FPGA hardware resources- our neural classifier employs 10 multipliers and can manage 1 coefficient per cc.

Haar DWT Preprocessing

Computationally, Haar Transform is a very simple DWT since its 1-D filters are: L=[1/2, 1/2] and H=[1/2, -1/2]. Therefore, any coefficient $\boldsymbol{H\_LL}_2(i, j)$ can be computed in one step according to:

$$\boldsymbol{H\_LL}_2(i,j) = \frac{1}{16}\sum_{l=0}^{l=3}\sum_{k=0}^{k=3} p(4i+k, 4j+l) \tag{15}$$

In order to compute (15), we exploit the same SHIFTREGISTERS block used for performing Daubechies DWT and a HAAR_LL2_FILTER block. HAAR_LL2_FILTER trivially adds[3] the data coming from OutToHaar16bytes[255..0] which are the values of the pixels $p(m, n)$ of the 4x4 window centered on the 16x16 sliding array implemented by SHIFTREGISTERS.
By this way, after a latency of 2 cc, HAAR_LL2_FILTER produces 1 coefficient (expressed by 12 bits) per cc and provides it to MLPN_CLASSIFIER via HaarLL2[11..00]. Higher performance is unnecessary, since the data flow of this block is parallel at that of

[3] The scaling by 16 is simply performed by a shift left of the fixed point of 4 positions.

DAUB_LL2_FILTER.

Multi Layer Perceptron Neural Classifier

As we have seen in Paragraph 4, the MLPN_CLASSIFIER implements two classifiers (DC and HC, see Figure 1) . Their structure is similar to that already described in Figure 15. The logical AND of their output is sent to the OUTPUT_INTERFACE via DCOutXHCOut.

Output Interface

The result of the classification is extended in a word of 64 bits by and sent to the host DataOut[63..0].

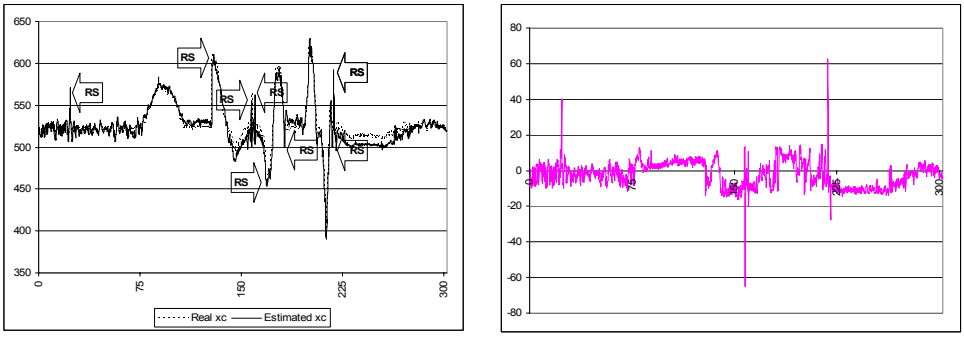
7. Experimental Results and Performance

In order to design and test ViSyR's processing core, a benchmark video sequence of more than 3,000,000 lines, covering a rail network of about 9 km was acquired. These were used in order to conduct several experiments aiming firstly at defining some methodological strategies and then at designing and testing the resulting system. In the following, several of the above experiments are described.

7.1 Rail Detection Methodologies Definition

Firstly, the approach to be used for the rail head detection algorithm has been selected comparing different approaches. In order to do this, methods based on Correlation, on Gradient based neural network, on PCA with threshold, PCA with neural network classifier, were implemented in software. A subset of the benchmark video sequence was sampled at a rate of 1000 lines, taking care of including among them, several lines showing rail switches. The obtained vector, of more than 300 lines, was manually inspected, detecting the real value of x_c, to be used as reference in order to evaluate the precision reachable by the tested methods. Among those, PCA with neural network classifier resulted the most accurate.

In Figure 18 are reported the coordinates of x_c both real (i.e., manually extracted) and automatically estimated by the realized system. The average of the absolute error was 6.04 pixels. The only evident discontinuities occur in concomitance of three rail switches, resulting in the spikes of Figure 18.b which reports the magnified error. We would put in evidence that, five other switches have been correctly analyzed. Anyway, except in these cases, the errors are almost always less than 10 pixels, and never more than 20. This error makes the method fully efficient for our practical purpose.



(a) (b)

Figure 18. (a): Real and estimated coordinates of x_C . (b): error. RS denotes rail switch

7.2 Single Value Decomposition Matrices Construction Definition

Matrices **A** and **C** were derived according to (A.1) and (A.4) using 450 examples of vectors $\mathbf{r}_i$ extracted from the acquired video sequence. After having determined the eigenvectors $\mathbf{u}_j$ and their eigenvalues λ_j, we verified that 12 eigenvectors were enough to represent the 91% of the information content of input data.

7.3 MLPN Classifiers Training Value

Error Back Propagation algorithm with an adaptive learning rate [Bishop (1995)] was used to determine the biases and the weights of the PCAC classifier. The adopted training set contained 262 different 400-pixels vectors centered on the rail (positive examples) and 570 negative examples consisting of 400-pixels vectors extracted from the video sequence, for what concerned RD&TB, while, for BDB, 391 positive examples of hexagonal-headed bolts with different orientations, and 703 negative examples consisting of 24x100 pixels windows extracted from the video sequence were used.

7.4 Activation Function Design

The analytical hardware implementation of the activation function $f(x)$ -equation (3)- needs huge resources, as well as, introduces much latency. We have implemented it by a look up table AF_LUT, storing 4096 values $f(x')$ computed onto 4096 equidistant values in [-5, 5] and assuming:

$$f(x)=\begin{cases}\text{if} \quad x<-5 & : \quad 0 \\ \text{if} \quad -5\le x\le 5 & : \quad \text{AF\_LUT}[x'] \qquad x' \quad \text{being the rounded value of } x \\ \text{if} \quad x>5 & : \quad 1\end{cases} \tag{16}$$

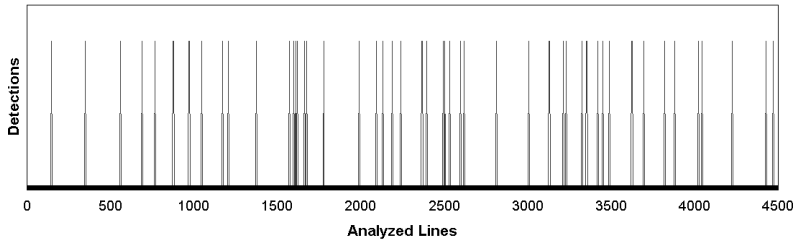
AF_LUT was filled using words of 5 bits, that was found the best compromise in terms of detection accuracy and hardware cost.

7.5 False Positive Elimination

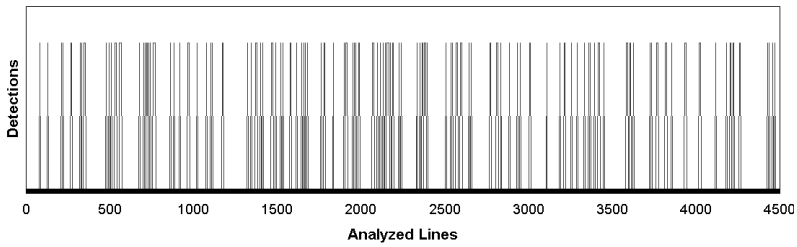
In defining the preprocessing strategy, we observed that, though the classifier DC, based on Daubechies DWT, reached a very high detection rate (see Paragraph 7.9), it also produced a certain number of False Positives (FPs) during the Exhaustive search.

In order to reduce these errors, a "cross validation" strategy was introduced. Because of its very low computational overhead, Haar DWT was taken into account and tested. HC, a neural classifier working on the $\boldsymbol{LL}_2$ subband of the Haar DWT, was designed and trained: HC reached the same detection rate of DC, though revealing much more FPs.

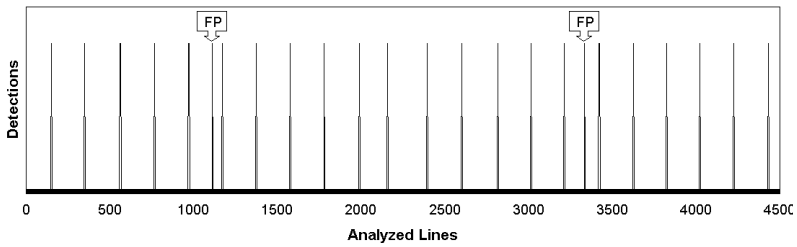
Nevertheless, the FPs resulting from HC were originated from different features (windows) than those causing the FPs output from DC. This phenomenon is put in evidence by Figure 19, where a spike denotes a detection (indifferently true and false positives) at a certain line of the video sequence revealed by DC (Figure 19.a) and by HC (Figure 19.b) while they analyzed in Exhaustive search (i.e., without jump between couple of bolts) 4,500 lines of video sequence. Figure 19.c shows the logical AND between the detections (both True and False Positive) of DC and HC. In other words, it shows the results of (7).



(a)



(b)



(c)

Figure 19. Detected couples of bolts vs video sequence, analyzed in Exhaustive search (i.e., without jump between couples of detected bolts). (a) Daubechies Classifier; (b) Haar Classifier; (c) Crossed validation

	True Positive (TP)	False Positive (FP)	FP/TP	FP/Analyzed Lines
Haar DWT	22 (100%)	90	409%	200.0‰
Daubechies DWT	22 (100%)	26	118%	57.8‰
AND (Daubechies, Haar)	22 (100%)	2	9%	4.4‰

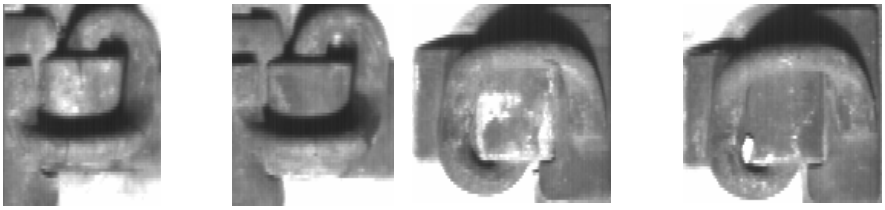
Table 1. False Positive (Exhaustive Search)

As it is evidenced, only 2 FPs over 4,500 analyzed lines (90,000 processed features) are revealed by the crossed validation obtained by the logical AND of DC and HC. Numerical results are reported in Table 1.

It should be noted that the shown ratio FP/TP is related to the Exhaustive search, but it strongly decreases during the Jump search, which skips a large number of lines that of course do not contain bolts.

7.6 Hook Bolts Detection

In order to test the generality of our system in detecting other kinds of bolts, we have tested ViSyR even on the hook bolts. Firstly, a second rail network employing hook bolts (see Figure 20) and covering about 6 km was acquired.



(a) (b)

Figure 20. Sample image patterns of the (a) right hook bolts and (b) left hook bolts

Two training sets TS1 and TS2 were extracted. They contained 421 negative examples, and respectively 172 positive examples of left hook bolts (TS1), and 172 examples of right hook bolts (TS2). Therefore, TS1 and TS2, were used for training the MLPN Classifiers devoted to inspect respectively the left and on the right side of the rail. Finally, the remaining video sequence was used to test the ability of ViSyR even in detecting hook bolts.

7.7 Hardware Design Definition

The report (file log) obtained from the above experiment was used as term of comparison for the reports of similar experiments aiming at defining the number of bits per words to be used in the hardware design. The fully-software prototype of ViSyR was modified changing the floating point operating mode into the fixed point mode. Different versions of ViSyR were compiled with different precisions (i.e., number of bits). For what concerned RD&TB, 12 bits for the eigenvectors coefficients and 28 bits for the weights of the classifier, allowed an accuracy only 0.6% lower than that one achievable using floating point precision while 23 bits for the filter coefficients and with 25 bits for the weights of both the classifiers led to detect visible bolts with accuracy only 0.3% lower than that obtained using floating point precision. These settings were considered acceptable, and the hardware design was developed using these specifications.

7.8 Rail Corrugation Analysis and Classification Strategy

As said in Paragraph 5, feature vectors have been respectively determined considering mean and variance of:

- each Gabor filter output image $i_\theta(x,y)$, one for orientation θ (0, π/2, π, ¾ π), getting a feature vector composed by 8 features;
- each HL, LH and HH subbands of each decomposition level, getting a feature vector composed by 18 features;
- each image of the jet (consisting of three decomposition levels -as in the wavelet transform case- per four orientations -as in the Gabor Filter case-), getting a feature vector composed by 24 features.

In order to test the performances of a k-Nearest Neighbor classifier, we have used a leave-one-out (LOO) procedure. Table 2 shows the number of misclassifications for different

values of ***K***, for a training set of Gabor filtered images (GF), Wavelet filtered images (WF) and Gabor-Wavelet filtered images (GWF).

	K						
	3	5	7	9	11	13	15
GF	3	3	6	5	5	4	5
WF	3	4	10	13	14	14	16
GWF	3	5	4	5	5	4	5

Table 2. KNN Classifier: Number of misclassifications for different values of K

In order to make independent the results from the kind of classifier, we have performed a comparison with the SVM classifier. In a preliminary step, we have evaluated the optimal regularization parameter C and polynomial kernel K(x,y) in order to configure the SVM classifier and get the best performance in terms of accuracy for the whole system. The results, using the LOO procedure, are presented in Table 3 for a regularization parameter C=150 and a polynomial kernel $K(x,y)=[(xy)/k]$ where k is a normalization factor for the dot product.

	C=150, K(x,y)=[(xy)/k]
GF	0
WF	12
GWF	10

Table 3. SVM Classifier: Number of misclassifications for C=150 and K(x,y)=[(xy)/k]

7.9 Accuracy and Computing Performance

The accuracy of RD&TB was measured on a test set of more than 1,500 vectors (832 positives i.e., rails, 720 negatives i.e., non rails). 99.8% of positives and 98.2% of negatives were correctly detected. The accuracy in detecting the presence/absence of bolts was also measured. A fully-software prototype of ViSyR, employing floating point precision, was executed in "trace" modality in order to allow an observer to check the correctness of the automatic detections. This experiment was carried out over a sequence covering 3,350 bolts. ViSyR detected 99.9% of the visible bolts, 0.1% of the occluded bolts and 95% of the absences. These performances have been possible also thanks to the crossed classification strategy described in Paragraph 4.

Even more accurate was the recognition rate in case of hook bolts, since together with a 100% of detected absent and present bolts, the system also achieved an acceptable rate detection of partially occluded hook bolts (47% and 31% respectively for left and right), whereas, it was not so affordable in case of occluded hexagonal bolts. This circumstances is justified since the hexagonal shape could cause miss classification because its similarity with the stones on the background.

Moreover, a better behavior in terms of detection of occluded hook bolts even speeds up the velocity. In fact, though the velocities reached during the Jump and the Exhaustive search does not present significant differences with respect those obtained with the hexagonal bolts the system remains (in the case of hook bolts) for longer time intervals in the Jump search, because of the higher detection rate. This leads to a higher global velocity.

For what concerns DAB, the comparative study aiming at define the most accurate feature extractor-classifier paradigm, it was found that a SVM classifier with C=150 and

K(x,y)=[(xy)/k], cascaded to a Gabor Filter, as described in Paragraph 5 reached 100% of detection both of corrugated and non-corrugated rails.
Table 4 resumes ViSyR's accuracy.

		Detection Rate
RD&TB	rail vectors	99.8%
	non-rail vectors	98.2%
BDB	visible hexagonal bolts	99.6%
	occluded hexagonal bolts	0.1%
	absent hexagonal bolts	95%
	visible left hook bolts	100%
	occluded left hook bolts	47%
	absent left hook bolts	100%
	visible right hook bolts	100%
	occluded right hook bolts	31%
	absent right hook bolts	100%
DAB	corrugated rails	100%
	non-corrugated rails	100%

Table 4. Detection accuracy

Computing performance was measured too, for what concerns the functionality of RD&TB and BDB (i.e. the ViSyR's modules already implemented in hardware). In particular, over than 15,000 couples of bolts have been detected in more than 3,000,000 lines at the velocity of 166 km/h. This performance is given by the combination of the Jump search and of the Exhaustive search, being the velocities reached during these phases approximately of 4 km/h and 444 km/h, and obviously depends on the distribution of the two kinds of search for the inspected video sequence. For instance, Figure 21 shows how the two types of search commutate during the process, for the tested video sequence.

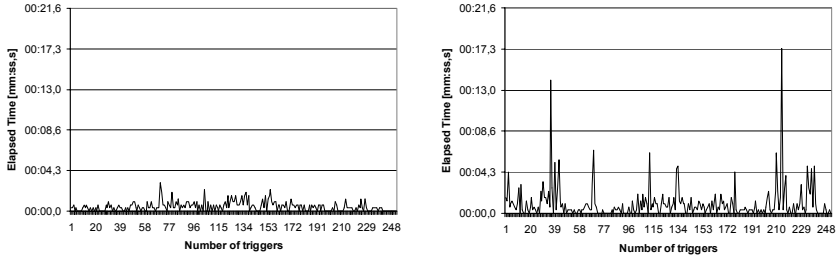


Figure 21. The way in which the system commutates during (a) the Exhaustive search and (b) the Jump search

The maximum elapsed time in the Exhaustive search is less than 3". This means that the Exhaustive search finds a couple of bolts (left and right) after less than 3" in the worst cases. At this point the control switches on the Jump search that, because of its philosophy, is much faster. When activated, Jump search works uninterruptedly up to 17", for the analyzed sequence (Figure 21.b). Obviously, if the system remains in the Jump phase for a long time, performance can increase subsequently. Next work will be addressed in this

direction, for example, automatically skipping those areas where the fastening elements are covered by asphalt (i.e., level crossing, where Exhaustive search is executed in continuous).

8. Conclusive Remarks

This paper has presented ViSyR, a visual system able to autonomously detect the bolts that secure the rail to the sleepers and to monitor the rail condition.
Thanks to a FPGA-based hardware implementation, it performs its inspection at velocities that can reach 460 km/h. In addition to this computing power ViSyR is also characterized by an impressive accuracy and is highly flexible and configurable, being the decision level of both RD&TB, BDB and DAB based on classifiers that can be easily reconfigured in function of different type of rails and bolts to be inspected and detected.
ViSyR constitutes a significant aid to the personnel in the railway safety issue because of its high reliability, robustness and accuracy. Moreover, its computing performance allows a more frequent maintenance of the entire railway network.
A demonstrative video of ViSyR is available at:
http://ftp-dee.poliba.it:8000/Marino/ViSyR_Demo.MOD

9. References

Alippi C., Casagrande E., Scotti F., & Piuri V. (2000) Composite Real-Time Image Processing for Railways Track Profile Measurement, *IEEE Trans. Instrumentation and Measurement,* vol. 49, N. 3, pp. 559-564 (June 2000).

Antonini M., Barlaud M., Mathieu P. & Daubechies I. (1992). Image Coding Using Wavelet Transform, *IEEE Trans. Image Processing,* Vol. 1, pp. 205-220. (1992).

Bahlmann C., Haasdonk B. & Burkhardt H. (2002). On-line Handwriting Recognition using Support Vector Machines - A kernel approach, *In Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)* 2002, Niagara-on-the-Lake, Canada (August 2002).

Benntec Systemtechnik Gmbh, RAILCHECK: image processing for rail analysis, *internal documentation*, http://www.benntec.com

Bishop M. (1995). *Neural Networks for Pattern Recognition*, New York, Oxford, pp. 164-191.

Bovik AC, Clark M, Geisler WS (1990), Multichannel texture analysis Using Localized Spatial Filters. *IEEE Trans On PAMI* 12: 55-73

Coreco. http://www.coreco.com

Cybernetix Group (France), IVOIRE: a system for rail inspection, *internal documentation*, http://www.cybernetix.fr

Daubechies I. (1988). Orthonormal bases of compactly supported wavelets, *Comm. Pure & Appl. Math.*, vol. 41, pp. 909-996. (1988).

Daubechies I. (1990 a). The Wavelet Transform, Time Frequency, Localization and Signal Analysis, *IEEE Trans. on Information Theory*, vol. 36, n. 5, pp. 961-1005. (Sept. 1990).

Daubechies I (1990 b), *Ten Lectures on Wavelets. Capital City Press, Montpellier*, Vermont

Drucker H., Burges C., Kaufman L., Smola A., Vapnik V. (1997). "Support Vector Regression Machines," in: M. Mozer, M. Jordan, andT. Petsche (eds.), *Neural Information Processing Systems*, Vol. 9. MIT Press, Cambridge, MA.

Gong S. et al. (2001). *Dynamic Vision: From Images to Face Recognition*, Imperial College Press.

Jain A., Duin R., & Mao J. (2000). Statistical Pattern Recognition: A Review, *IEEE Transactions on Pattern Analysis and Machine Intelligenve,* vol. 22, no.1, pp.4-37, 2000.

Jain AK, Farrokhnia F (1990). *Unsupervised texture segmentation using Gabor filters. Pattern Recognition*, 24: 1167-1186

Lee T.S. (1996). Image Representation Using 2D Gabor Wavelets , *IEEE Trans. on PAMI* , Vol. 18 no. 10, 1996

Ma W. Y., Manjunath B.S. (1995) A comparison of wavelet transform features for texture image annotation, *Proc. Second International Conference on Image Processing (ICIP'95),* Washington, D.C., vol. 2, pp. 256-259. (Nov. 1995).

MachineVision. CAMERALINK: specification for camera link interface standard for digital cameras and frame grabbers, www.machinevisiononline.org

Mallat S.G. (1989). A theory for quadriresolution signal decomposition: the wavelet representation, *IEEE Trans on Pattern Analysis and Machine Intelligence*, vol. 2 pp.674-693 (1989).

Matrox. http://www.matrox.com/imaging/products/odyssey_xcl/home.cfm

Mazzeo P.L., Nitti M., Stella E. & Distante A. (2004). Visual recognition of fastening bolts for railroad maintenance, *Pattern Recognition Letters*, vol. 25 n. 6, pp. 669-677 (2004).

Osuna E., Freund R. & Girosi F. (1997) Training Support Vector Machines: an Application to Face Detection, *Proceedings of CVPR'97*, Puerto Rico. (1997).

Papageorgiou C. & Poggio T. (1999). A Pattern Classification Approach to Dynamical Object Detection, *Proceedings of ICCV*, pp. 1223-1228 (1999).

Porat M, Zeevi YY (1988), The generalized Gabor scheme of image representation in biological and machine vision, *IEEE Trans Pattern Anal Machine Intell* 10: 452-468

Rubaai A. (2003). A neural-net-based device for monitoring Amtrak railroad track system, *IEEE Transactions on Industry Applications*, vol. 39, N. 2 , pp. 374-381 (March-April 2003).

Sato K., Arai H., Shimuzu T., & Takada M. (1998). Obstruction Detector Using Ultrasonic Sensors for Upgrading the Safety of a Level Crossing, *Proceedings of the IEE International Conference on Developments in Mass Transit Systems*, pp. 190-195 (April 1998).

Stella E., Mazzeo P.L., Nitti M., Cicirelli G., Distante A. & D'Orazio T. (2002). Visual recognition of missing fastening elements for railroad maintenance, *IEEE-ITSC International Conference on Intelligent Transportation System*, pp. 94-99, Singapore (2002).

Strang G., & Nuguyen T. (1996). *Wavelet and Filter banks*, Wellesley College.

Vapnik N. (1998), *Statistical Learning Theory*, New York: John Wiley & Sons Inc. Pub.

Wen J, Zhisheng Y, Hui L (1994), Segment the Metallograph Images Using Gabor Filter, *International Symposium on Speech Image Processing and Neural Networks* pp 25-28, Hong Kong

Xishi W., Bin N., & Yinhang C. (1992). A new microprocessor based approach to an automatic control system for railway safety, *Proceedings of the IEEE International Symposium on Industrial Electronics*, vol. 2, pp. 842-843 (May 1992).

Yinghua M., Yutang Z., Zhongcheng L., & Cheng Ye Y. (1994). A fail-safe microprocessor-based system for interlocking on railways, *Proceedings of the Annual Symposium on Reliability and Maintainability*, pp. 415-420 (Jan. 1994).

Appendix A. Principal Component Analysis (PCA)

Let $\mathbf{i}_j$ row-images, each one having N pixels, object of the analysis.
Let R a set of P images $\mathbf{r}_k$ (k=1..P, $P \geq N$). Such images $\mathbf{r}_k$, having Q pixels with Q <N, have been extracted from the images $\mathbf{i}_j$, and chosen in order to select instances of the objects.

Figure 22. Rail head row image example

Let **A** the Q rows and P columns matrix:

$$\mathbf{A}=[\mathbf{h}_1 ,\ldots., \mathbf{h}_P] \tag{A.1}$$

with:

$$\mathbf{h}_k = \mathbf{r}_k - \boldsymbol{\mu} \tag{A.2}$$

where:

$$\boldsymbol{\mu}= [\mu_1,..,\mu_P]^T \tag{A.3}$$

with μ_k denoting the average of intensities in $\mathbf{r}_k$.
From **A**, the covariance matrix:

$$\mathbf{C}=\mathbf{A}\mathbf{A}^T \tag{A.4}$$

can be built. The QxQ matrix $\boldsymbol{C}$ contains information about mutual relationships among rail images $\mathbf{r}_k$.
In Principal Component Analysis [Gong *et al.* (2001), Jain *et al.* (2000).] the eigenvectors $\mathbf{u}_j$ (j=1..N) of **C** define a new reference space in which the variance among data is maximized. Moreover, an ordering relationship on $\mathbf{u}_j$ components can be induced sorting the eigenvectors $\mathbf{u}_j$ in such way that:

$$\lambda_q > \lambda_{q+1} \quad (q=1, .., Q-1) \tag{A.5}$$

where the eigenvalues λ_j of **C**, represent the variances of each one of $\mathbf{u}_j$. In other words, (A.5) means that the set of projections of the input data on $\mathbf{u}_q$ has variance higher than that one of the set of projections of the input data on $\mathbf{u}_{q+1}$.
By thresholding the eigenvalues λ_j it is possible to select the corresponding L<Q eigenvectors sufficient enough to represent the biggest part of the informative content of the input data. Let λ_l (l=1..L, L<Q) the selected components, a generic vector $\mathbf{r}'$ can be expressed by:

$$\mathbf{r}' \approx \sum_{l=1}^{L} a_l \mathbf{u}_l + \boldsymbol{\mu}' \tag{A.6}$$

where $\boldsymbol{\mu}'$ is the average vector of $\mathbf{r}'$. From a computational point of view the eigenvectors and eigenvalues of **C** can be estimated by a Single Value Decomposition (SVD) of matrix **A** where the coefficients a_l are evaluated by the inner product:

$$a_l = (\mathbf{r}'-\boldsymbol{\mu}')\mathbf{u}_l^T \tag{A.7}$$

In this scenario, the vector

$$\mathbf{a}'=[a_1,\ldots,a_L]^{\mathrm{T}} \tag{A.8}$$

can be considered a feature containing most of information content of $\mathbf{r}'$.

Appendix B. Gabor Filter

In the complex spatial 2D domain, Gabor filter is given by:

$$h(x,y)=g(x',y')\cdot e^{2\pi jFx'} \tag{B.1}$$

where

$$g(x,y)=\frac{1}{2\pi\sigma_x\sigma_y}\cdot e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} \tag{B.2}$$

and x' and y' are the rotated coordinates:

$$(x',y')=(x\cos\theta+y\sin\theta,-x\sin\theta+y\cos\theta) \tag{B.3}$$

σ_x and σ_y are the standard deviations of Gaussian envelope along the x and y directions, F frequency of sinusoidal plane and and θ is the orientation [Wen at al. (1994)].
Thus (B.1) is a complex sinusoidal grating modulated by a 2D gaussian function [25].
Gabor functions have been found useful because reach the lower bounds of the uncertainty inequalities $\Delta x\Delta u\geq 1/4\pi$ and $\Delta y\Delta v\geq 1/4\pi$ and achieve optimally joint resolution in space and spatial frequency [Bovik et al. (1990)].

Appendix C. Wavelet Transforms

The wavelet transform [Daubechies (1988), Mallat (1989), Daubechies (1990 a), Antonini *et al*. (1992)], is a mathematical technique that decomposes a signal in the time domain by using dilated/contracted and translated versions of a single finite duration basis function, called the prototype wavelet. This differs from traditional transforms (e.g., Fourier Transform, Cosine Transform, etc.), which use infinite duration basis functions. One-dimensional (1-D) continuous wavelet transform of a signal $x(t)$ is:

$$W(a,b)=\frac{1}{\sqrt{a}}\int x(t)\overline{\psi}\left(\frac{t-b}{a}\right)dt \tag{C.1}$$

where $\overline{\psi}\left(\frac{t-b}{a}\right)$ is the complex conjugate of the prototype wavelet, $\psi\left(\frac{t-b}{a}\right)$; a is a time dilation and b is a time translation.
Due to the discrete nature (both in time and amplitude) of most applications, different Discrete Wavelet Transforms (DWTs) have been proposed according to the nature of the signal, the time and the scaling parameters.
The two-dimensional (2-D) DWT works as a multi-level decomposition tool. A generic 2-D DWT decomposition level j is shown in Figure 23.

It can be seen as the further decomposition of a 2-D data set $\boldsymbol{LL}_{j-1}$ ($\boldsymbol{LL}_0$ being the original input image) into four subbands $\boldsymbol{LL}_j$, $\boldsymbol{LH}_j$, $\boldsymbol{HL}_j$ and $\boldsymbol{HH}_j$. The capital letters and their position are related respectively to the applied mono-dimensional filters (*L* for Low pass filter, *H* for High pass filter) and to the direction (first letter for horizontal, second letter for vertical). The band $\boldsymbol{LL}_j$ is a coarser approximation of $\boldsymbol{LL}_{j-1}$. The bands $\boldsymbol{LH}_j$ and $\boldsymbol{HL}_j$ record the changes along horizontal and vertical directions of $\boldsymbol{LL}_{j-1}$, respectively, whilst $\boldsymbol{HH}_j$ shows high frequency components. Because of the decimation occurring at each level along both the directions, any subband at the level *j* is composed by N_jxM_j elements, where $N_j=N_0/2^j$ and $M_j=M_0/2^j$.

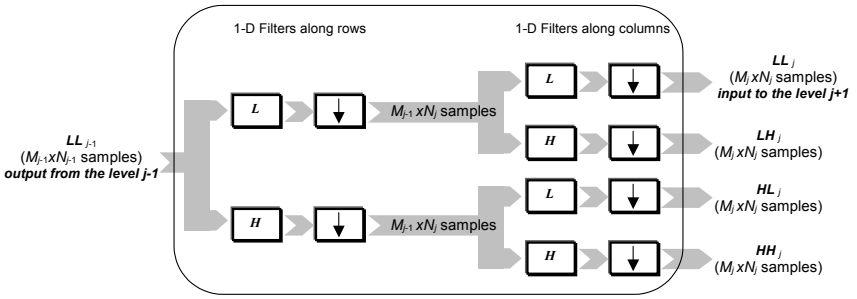


Figure 23. 2-D DWT: The j^{th} level of subband decomposition. ↓ represents decimation by 2

Appendix D. Gabor Wavelet Transform

As seen in Appendix C, Wavelet transform can be chosen as mathematical model for its adaptability in resolution both in frequency and space domains relating to a scale parameter, while Gabor filters assure the lower limits of uncertainty inequalities (as described in Appendix B) in the space frequency domain. As consequence, Gabor functions can be considered as mother function of the Wavelet transform. On these bases, a set of 2D Gabor Wavelet filters can be defined through a projection of the signal into a family of *M* Gabor Wavelet functions $\Psi=\{\psi_{n_1},\psi_{n_2},\ldots,\psi_{n_M}\}$ derived from a process of contractions and dilations of a function, the so-called *mother Gabor-Wavelet*.
In two dimensions the Gabor Wavelet Functions [Lee (1996)] take the form:

$$\psi_{\mathbf{n}}(x,y)=\frac{s_x^{1/2}}{2\pi\sigma^2}e^{-\frac{1}{2\sigma^2}\left\{\left[s_x\left((x-c_x)\cos\theta+(y-c_y)\sin\theta\right)\right]^2+\left[s_y\left(-(x-c_x)\sin\theta+(y-c_y)\cos\theta\right)\right]^2\right\}}e^{2\pi jFs_x\left((x-c_x)\cos\theta+(y-c_y)\sin\theta\right)} \quad \text{(D.1)}$$

where **n** is a parametric vector $[c_x,c_y,\theta,s_x,s_y]$, with c_x and c_y representing the contractions of the GWT along *x* and *y* respectively, s_x and s_y represent the dilations along the two scales, and θ the orientation.
In addition, the dilations s_x and s_y can be selected as $s_x=s_y=2^l$ for $l=0,\ldots,L$-1, with *L* is the number of decomposition levels, and $s_x\,c_x=s_y\,c_y=k$. As consequence, (D.1) can be written as:

$$\psi_{\mathbf{n}}(x,y)=\frac{s_x^{l/2}}{2\pi\sigma^2}e^{-\frac{1}{2\sigma^2}\left\{\left[2^l\left((x-k2^{-l})\cos\theta+(y-k2^{-l})\sin\theta\right)\right]^2+\left[2^l\left(-(x-k2^{-l})\sin\theta+(y-k2^{-l})\cos\theta\right)\right]^2\right\}}e^{2\pi jF2^l\left[(x-k2^{-l})\cos\theta+(y-k2^{-l})\sin\theta\right]} \quad \text{(D.2)}$$

and the responses of Gabor-Wavelet filters $i_{l,\mathrm{n}}(x,y)$ can be defined as:

$$i_{l,\mathbf{n}}(x,y)=\psi_{\mathbf{n}}(x,y)*i(x,y) \tag{D.3}$$

where *l* is a certain level into pyramidal structure.

Appendix E. Support Vector Machine (SVM)

Support Vector Machine (SVM) [Vapnik (1998)] is based on the structural risk minimization principle from computational learning theory, or better on minimization of the misclassification probability of vectors with unknown distribution of data. With respect to the neural approach, SVM allows a better control of dynamics of the classifier. Examples of use of the SVM are given in [Bahlmann et al. (2002), Papageorgiou & Poggio. (1999) Drucker *et al.* (1997), Osuna *et al.* (1997)]. The basic idea of SVM consists of imagining some hyper-planes that divide the hyper-space containing the vectors **v** to be classified into two sub-hyper-spaces where positive examples of **v** (classified with +1) and negative examples of **v** (classified with -1) of the training set $S=\{\mathbf{v}_1,\mathbf{v}_2,\ ...,\mathbf{v}_N\}$ are respectively located.

There are many possible classifiers that can separate the data with hyper-planes $\mathbf{w}\cdot\mathbf{v}+b=0$, but there is only one that maximizes the distance between the closest vectors to the hyper-plane and the hyper-plane itself. SVM finds the optimal separating hyper-plane:

$$\mathbf{w}^*\cdot\mathbf{v}+b^*=0 \tag{E.1}$$

maximizing the margin and minimizing the number of misclassified patterns. In (E.1), the optimal weight vector is expressed as linear combination of the examples of the training set S:

$$\mathbf{w}^*=\sum_{i=1}^{N}\lambda_i^* y_i\mathbf{v}_i \tag{E.2}$$

where $y_i\in\{-1,\ 1\}$ is the label (or class) of the vector $\mathbf{v}_i$, and the optimum $\lambda^*=\{\lambda_1^*,\lambda_2^*,\ ...,\lambda_N^*,\}$ (where $\lambda_i^*\geq 0$) is a solution of a quadratic problem. The vectors $\mathbf{v}_i$ with $\lambda_i^*>0$ are said "support vectors". The classification of new vectors **v** involves the evaluation of the decision function *y=sign*(*f*(**v**)) where:

$$f(\mathbf{v})=\mathbf{w}^*\cdot\mathbf{v}+b=\left(\sum_{i=1}^{N}\lambda_i^* y_i\mathbf{v}_i\right)\cdot\mathbf{v}+b^* \tag{E.3}$$

meaning that **v** can be classified by evaluating the dot product between **v** and some elements (support vectors) of the training set S.