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

Source: Vision Systems: Applications, ISBN 978-3-902613-01-1
Edited by: Goro Obinata and Ashish Dutta, pp. 608, I-Tech, Vienna, Austria, June 2007

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.

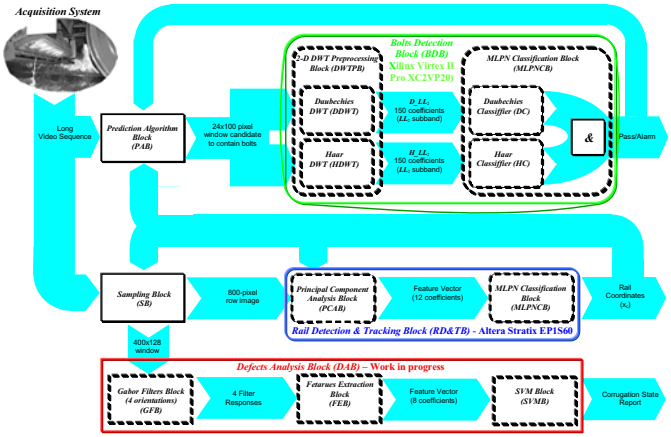


Figure 1. ViSyR's Functional diagram. Rounded blocks are implemented in a FPGA-based hardware, rectangular blocks are currently implemented in a software tool on a general purpose host

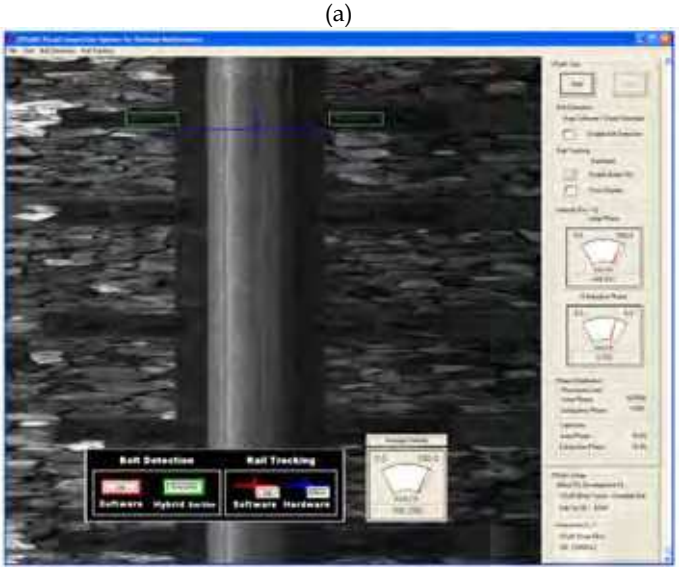
RD&TB employs PCA followed by a Multilayer Perceptron Network Classification Block (MLPNCB) for computing the coordinates of the center of the rail. More in detail, a Sampling Block (SB) extracts a row of 800 pixels from the acquired video sequence and provides it to the PCA Block (PCAB). Firstly, a 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 $\mathbf{a}'$ -determined as shown in (A.8)- in order to discriminate if it derives from a vector $\mathbf{r}'$ 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.

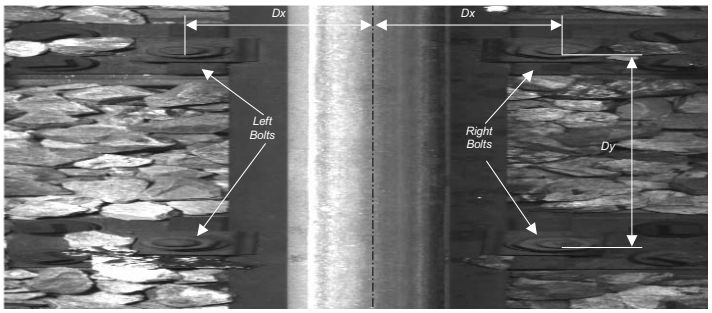


Figure 5. Geometry of a rail. A correct expectation for *Dx* and *Dy* notably reduces the computational load

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.

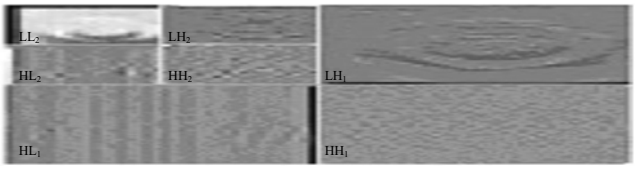


Figure 7. Application of two levels of 2-D DWT on a subimage containing an hexagonal-headed bolt

[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:

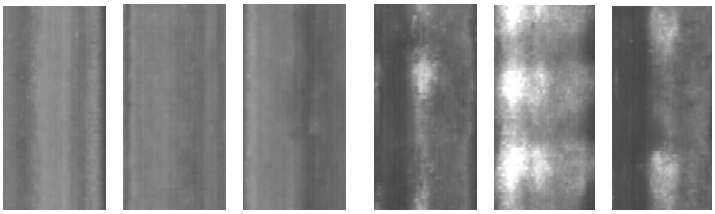
$$\text{Presence} = \left(D\_n'''_0 > 0.9\right) AND \left(H\_n'''_0 > 0.9\right) \tag{7}$$

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.

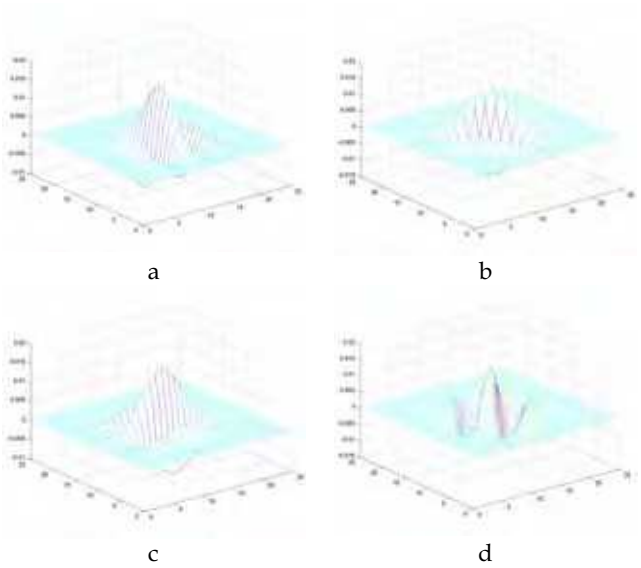


Figure 9. Gabor Filters at different orientations: (a) 0; (b) π/4; (c) π/2; (d) 3π/4

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:

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

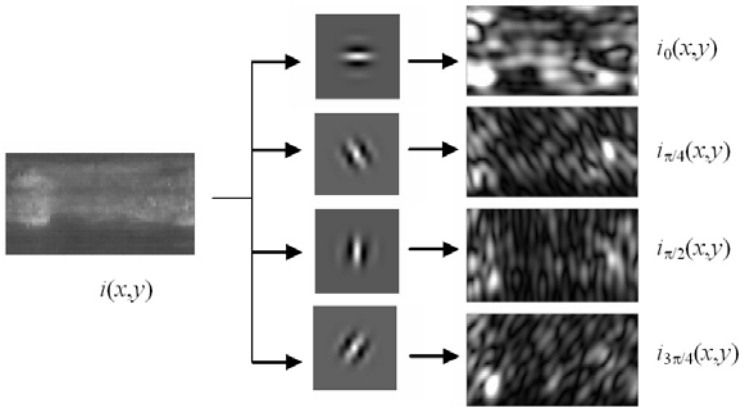


Figure 10. Examples of Gabor Filters ($F=\sqrt{2}/2^3$, σ=2) applied to a corrugated image

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.

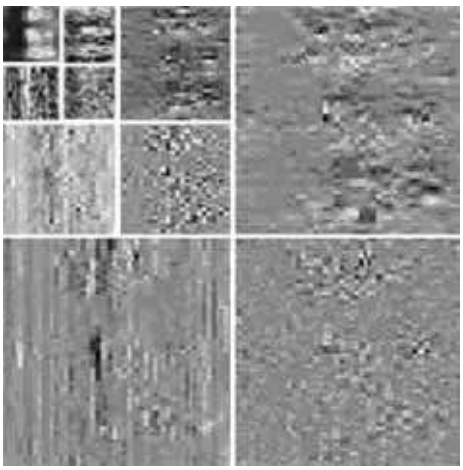


Figure 11. Example of "Daubechies 1" Discrete Wavelet transform (three decomposition levels) of the corrugated image

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.

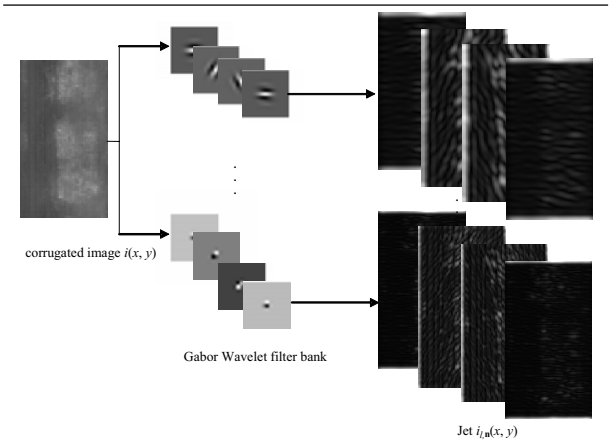


Figure 12. Example of Gabor Wavelet transform of the corrugated image

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:

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

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

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.

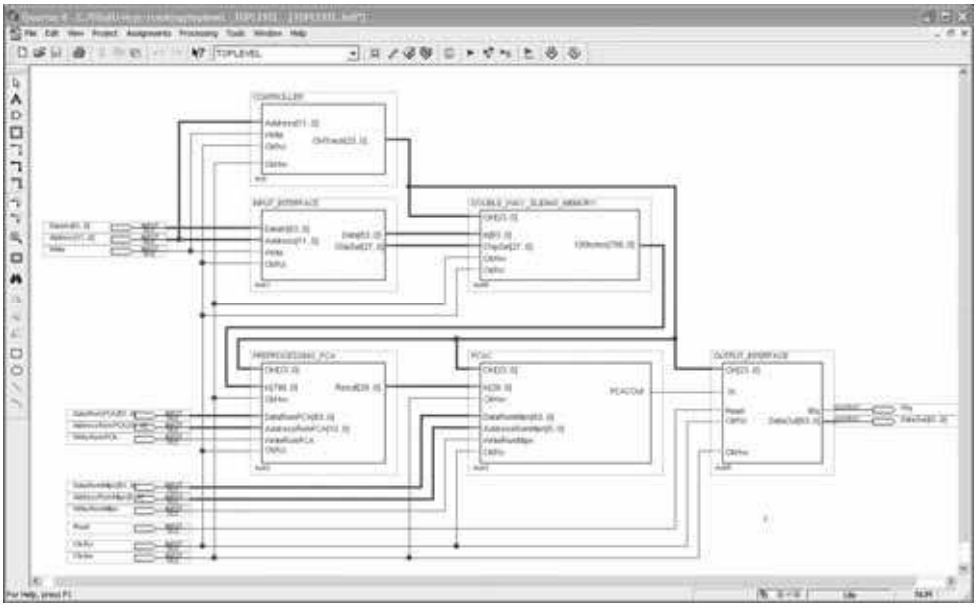
5.2 Classification

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.

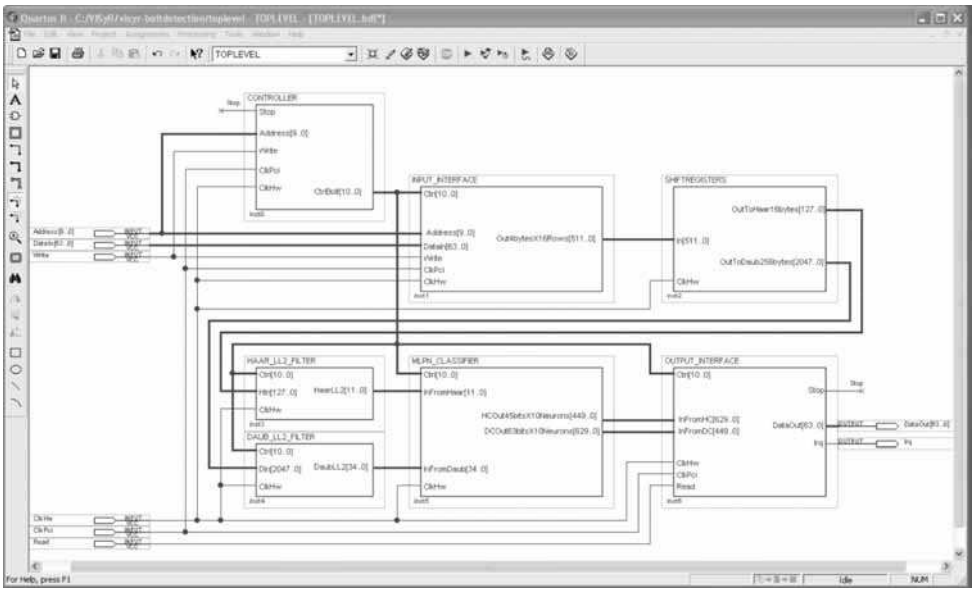
6. FPGA-Based Hardware Implementation

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.



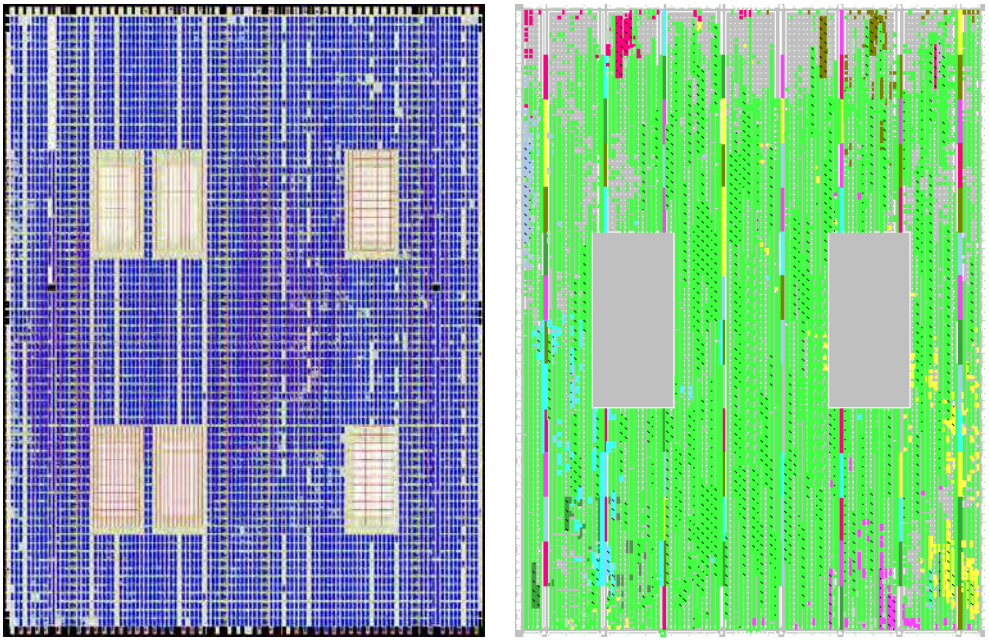
(a)



(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.

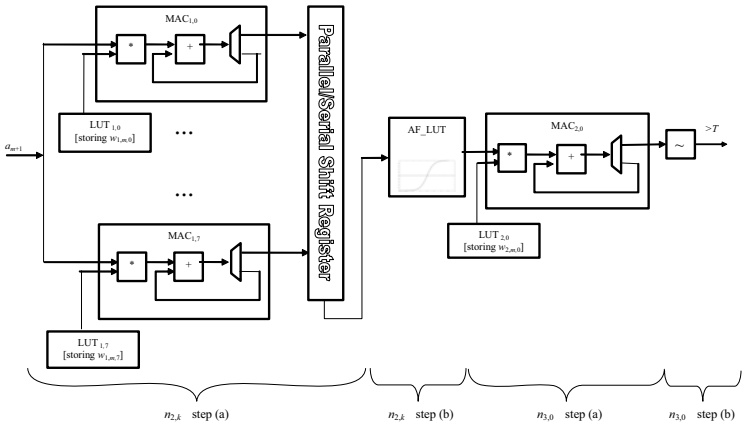


Figure 15. PCAC functionality

Specifically, step (a):

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