A GPU-BASED VISION SYSTEM FOR REAL TIME DETECTION OF FASTENING ELEMENTS IN RAILWAY INSPECTION

P. De Ruvo[1], *A. Distante*[1], *E. Stella*[1], *and F. Marino*[2]

[1] Istituto di Studi sui Sistemi Intelligenti per l'Automazione (ISSIA) CNR, ITALY
[2] Dipartimento di Elettrotecnica ed Elettronica (DEE), Politecnico di Bari, ITALY

ABSTRACT

The railway maintenance is a particular application context required in order to prevent any dangerous situation.

With the growing of the high-speed railway traffic, automatic inspection systems able to detect rail defects, sleepers' anomalies, as well as missing fastening elements, become strategic since they could increase the ability in the detection of defects and reduce the inspection time in order to guarantee more frequent maintenance of the railway network.

This paper presents a patented fully automatic and configurable real-time vision system able to detect the presence/absence of the fastening bolts that fix the rails to the sleepers. It gets an accuracy of 99.9%, and, thanks to its parallel processing allowed by a Graphic Processing Unit, reaches an average throughput of 187 km/h, speeding up of about 287 % the performance of a quadcore CPU implementation.

Index Terms—Visual Inspection, Pattern Recognition, General Purpose Graphical Processing Unit.

1. INTRODUCTION

The railway maintenance is a particular application context usually 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, since its results are strictly dependent on the observer's capability in detecting possible anomalies and in recognizing critical situations.

Even if some semi-automatic tools have been proposed (such as track profile measurement [1], obstruction detection [2], braking control [3], etc.), at the best of our knowledge, the only found approaches to the fastening elements recognition are commercial vision systems [4] which consider only fastening elements having shape with regular geometry, being bounded by their geometrical approaches 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 kind of fastening element is considered, the tuning phase has to be re-executed.

In this scenario, we propose a completely automatic system for the railway inspection and the fastening bolts detection which needs no tuning phase. The human operator has only the task of selecting images of the fastening elements to manage. Moreover, it requires no constraints on the shape of the fastening elements, being suitable for both geometric and generic shapes.

The inspection system is briefly presented in the next paragraph. A more detailed description of the bolts detection strategy is reported in paragraph 3. Paragraph 4 describes the computing environment and the parallel implementation. Accuracy and computing performance are reported in paragraph 5.

2. SYSTEM OVERVIEW

Our system acquires images of the rail by means of a DALSA PIRANHA 2 line scan camera [5] having 1024 pixels of resolution (maximum line rate of 67 kLine/s), provided with a PC-CAMLINK frame grabber (Imaging Technology CORECO) [6] and using the Cameralink protocol [7]. 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 making the system robust against changes in the natural illumination. The data acquisition is synchronized thanks to a wheel encoder triggering the line scan camera. By this way, independently from the train velocity, the resolution results along y (main motion direction) in 3 mm/pixel, and along the orthogonal direction x in 1 mm/pixel. The acquisition system has been installed under a diagnostic train.

The detection of the bolts is performed by a Prediction Module (PM) and a Bolt Detection Module (BDM) cooperating according the strategy described in paragraph 3. In doing this, they need the coordinate of the center of the rail-head x_c, that is detected and tracked by a Rail Detection and Tracking Module (RD&TM).

RD&TM does not operate in continuous, being called by the system only when the synchronism in founding the bolts fails and the system exits from the "jumping search" (see paragraph 3.1): in fact, a fault bolt detection indicates that the rail-head might need of being re-tracked. This module employs Principal Components Analysis followed by a Multi Layer Perceptual Network Classifier (MLNPC). Firstly, a vector of 400 pixels, extracted from a row of the video sequence and centered on the last detected x_c is multiplied by twelve different eigenvectors. The twelve coefficients generated by these scalar products are classified by a MLPNC which reveals if the processed vector is still centered on the rail head. In that case, the value of x_c is confirmed. Contrariwise, the module iterates the algorithm suitably shifting along x (alternately on left and on right) the previously processed vector, until a newly determined value can update x_c.

3. BOLTS DETECTION

Usually two kinds of bolts are used to secure the rail to the sleepers: hook bolts and hexagonal-headed bolts (Fig. 1).

They essentially differ by shape: the first one has a more complex hook shape and can be found oriented only in one

Fig. 1. Examples of fastening elements: hexagonal headed bolts, right hook bolts and left hook bolts. Resolutions along x and y are different because of the acquisition setup.

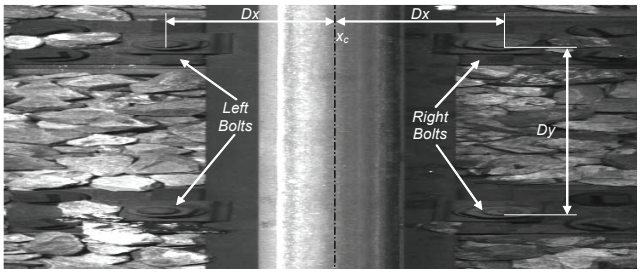

Fig. 2. Geometry of a rail. A correct expectation for x_c, Dx and Dy notably reduces the computational load.

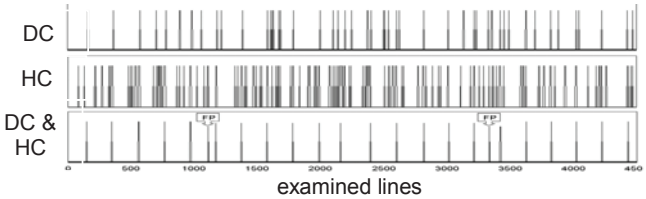

Fig. 3. Detected couples of bolts vs examined lines in a video sequence, when jumping search is disabled (i.e., analyzing indiscrimnately each line of the video sequence, without jumping between couples of detected bolts). From the top: Daubechies Classifier, Haar Classifier, double validation.

direction, the second one has a regular hexagonal shape with random orientation. Moreover, the hexagonal bolts allow geometrical approaches, but present similarity with the shape of the stones that are on the background, causing miss-classification.

Our system successfully detects both of them, since it is based on MLPNCs and consists of a prediction phase, identifying the image areas (frames) candidate to contain the patterns to be detected; a data reduction phase, reducing the computational load thanks to DWT; and a classification phase, revealing the presence/absence of the fastening elements.

3.1. Prediction Phase

In the rail structure, the distance Dx between rail and fastening elements is constant and *a priori* known. Similarly, the distance Dy between two adjacent couples of fastening elements is constant, though with a less precise approximation (see Fig. 2).

Exploiting this geometry, PM speeds-up the processing alternately using two kinds of search: "Exhaustive search" and "Jumping search".

In the first kind of search, a window exhaustively slides on the areas distant Dx from x_c until the first occurrence of the left and of the right fastening elements are detected "contemporaneously" (at the same y). At this point, the y coordinate of this couple of bolts (y_1) is stored, and the system may continue until it finds the second couple (coordinate y_2). Now, it sets $Dy = y_2 - y_1$ and switches the process on the Jumping search, using Dy in order to directly jump in those areas candidate to enclose the fastening elements, saving computational time. If, during the Jumping search, the fastening elements are not found in the position where they were predicted, then the system writes the position of the fault in a log-file, calls RD&TM for updating x_c, and restarts the Exhaustive search.

3.2. Data Reduction Phase

In pattern recognition, input images are generally pre-processed in order to extract their intrinsic features. For reducing the input space to the classifiers, we have found a features extraction algorithm able to concentrate all the important information on the input patterns in a small set of coefficients. This algorithm has been developed considering 2-D DWTs [8], since DWT concentrates the significant variations of input patterns in a reduced number of coefficients. We have tested different DWTs varying the number of decomposition levels, in order to reduce this set, without lose accuracy. Specifically, we have chosen both a compact wavelet introduced by Daubechies [8], and the Haar DWT (also known as Haar Transform, [9]) since we have verified that, for our specific application, the combined use of these two approaches avoids -almost completely- the false positive detection.

The best compromise has been reached by the Lowpass-Lowpass subband at the 2nd decomposition level (say, $\boldsymbol{LL}_2$) consisting only of 6x25 coefficients. Therefore, BDM computes $\boldsymbol{LL}_2$ of a Haar DWT and, if these are positively classified by HC (see paragraph 3.3), then BDM produces also $\boldsymbol{LL}_2$ of a Daubechies DWT, in order to perform a second validation. In fact, we have found that the double classification strategy described in the following paragraph, gets an accuracy of 99.9% in recognizing bolts in the primitive windows, practically at-all avoiding false positive detection.

3.3. Classification Phase

BDM employs two MLPNCs, say DC and HC. They were trained respectively for Daubechies DWT and Haar DWT using Error Back Propagation algorithm with an adaptive learning rate [Bishop (1995)] based on 391 positive examples of hexagonal-headed bolts with different orientations, and 703 negative examples consisting of 24x100 pixels windows extracted from the video sequence. DC and HC have an identical three-layers topology 150:10:1 and differ only for the weights.

To detect a presence of the bolt, the outputs from DC and HC are combined according to a logical AND. In fact, during the development of our system, we observed that though a discrimination based on Daubechies DWT reached a very high detection rate, it also produced a certain number of False Positives (FPs) during the Exhaustive search.

In order to reduce these errors, a "double validation" strategy was introduced. Because of its very low computational overhead, the Haar DWT was also taken into account designing and training HC, a neural classifier working on its $\boldsymbol{LL}_2$ subband. HC reached the same detection rate of DC, though revealing much more FPs. Nevertheless, we observed that the FPs resulting from HC were originated from different frames than those causing the FPs revealed by DC.

This phenomenon is put in evidence by Fig. 3, where a spike denotes a detection (indifferently true or false) at a certain examined line of the video sequence. As it is evidenced, only 2 FPs over 4,500 analyzed lines (90,000 processed frames) have resulted by the double validation obtained by the logical AND between DC and HC.

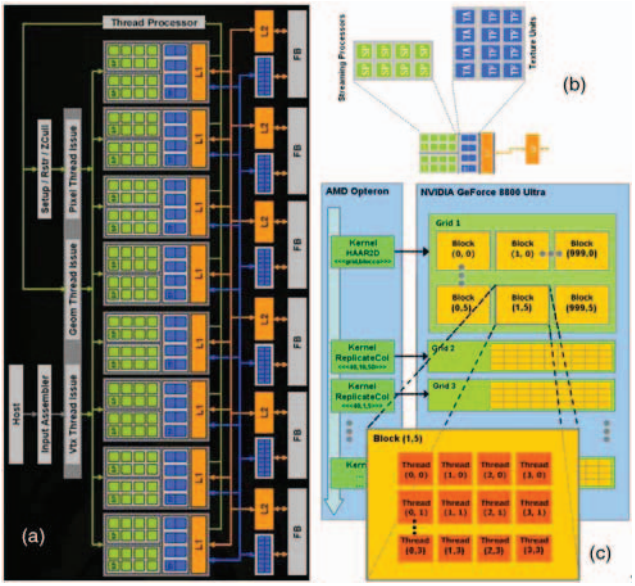

Fig. 4. Nvidia G80: Block Diagram (a); SPs and texture units (b); Thread batching of the described system (c).

```
// Initialization
cublasSetVector(800,sizeof(float),iSVDPatch,1,SVDPatch,1); // CL
SplitCol<<<401,401,2000>>>(SVDPatch,SVDPatchParte_L);      // CK
ReplicateCol<<<401,12,200>>>(FilterxMean,FilterxMean_L);   // CK
ReplicateCol<<<401,8,100>>>(HiddCoeff_bias,HiddCoeff_bias_L);// CK
ReplicateCol<<<401,1,30>>>(LastCoeff_bias,LastCoeff_bias_L);// CK
// Processing
cublasSgemm('n','n',12,401,401,1.0,Filtro_,12,SVDPatchParte_L,401,
                                -1.0,FilterxMean_L,12);   // CL
cublasSgemm('n','n',8,401,12,1.0,HiddenCoeff_,8,FilterxMean_L,12,1.0,
                                HiddenCoeff_bias_L,8);  // CL
Sigmoid<<<401,8,100>>>(HiddenCoeff_bias_L,ldata_L);   // CK
cublasSgemm('n','n',1,401,8,1.0,LastCoeff_,1,ldata_L,8,1.0,
                                LastCoeff_bias_L,1);    // CL
Sigmoid<<<401,1,30>>>(LastCoeff_bias_L,ldata_L); // CK
ris[0] = (double)cublasIsamax( 401 , ldata_L , 1 );   // CL
```

```
// Initialization
cublasSetVector(1500,sizeof(float),iHiddenCoeff,1,HiddenCoeff,1); // CL
cublasSetVector(10,sizeof(float),iLastCoeff,1,LastCoeff,1); // CL
cublasSetVector(4272,sizeof(float),iHAARPatch,1,HAARPatch,1); // CL
// Processing: HAAR
dim3 block(4,4);
dim3 grid(1000,6);
HAAR2D<<<grid,block>>>(HAARPatch,odata); // CK
// Processing: MLNPC
ReplicateCol<<<40,10,50>>>(HiddenCoeff_bias,
                                HiddenCoeff_bias_L); // CK
ReplicateCol<<<40,1,5>>>(LastCoeff_bias,LastCoeff_bias_L); // CK
cublasSgemm('n','n',10,40,150,1.0,HiddenCoeff,10,odata,150,
                                1.0,HiddenCoeff_bias_L,10); // CL
Sigmoid<<<40,10,50>>>(HiddenCoeff_bias_L,udata); // CK
cublasSgemm('n','n',1,40,10,1.0,LastCoeff,1,udata,10,
                                1.0,LastCoeff_bias_L,1); // CL
Sigmoid<<<40,1,5>>>(LastCoeff_bias_L,risultato); // CK
// Write Results
cublasGetVector(40,sizeof(float),risultato,1,Ris,1); // CL
```

Fig. 5. Extracted code from RD&TM (top) and BDM (down) modules. CL: CUBLAS Library; CK: CUDA Kernel.

4. GPU-BASED PARALLEL IMPLEMENTATION

The most relevant computational tasks of the system are those of RD&TM and BDM. Nevertheless, they mainly consist of algorithms (i.e. SVD, Haar and Daubechies DWTs, MLNPC), which are implicitly parallel and can easily exploit a SIMD implementation on dedicated hardware. Therefore, in order to reach a high throughput and real-time video analysis, we have adopted a NVIDIA 2-Way SLI, a device provided with two Graphical Processing Units (GPUs) GeForce 8800 Ultra, working in SLI (Scalable Link Interface) mode.

The GeForce 8800 Ultra [10], as shown in Fig. 4.a, contains 128 stream processors (also called shader processors, SPs) arranged into 8 clusters of 16 processors. The processors within the same cluster share the same L1 cache and (if required by the computation) may access the L1 cache of other clusters through a bus. The L2 cache is arranged into 6 partitions, each having a 64-bit interface to graphic memory, with a total width of 384 bits. Moreover, (see Fig. 4.b) each cluster of SP shares 4 texture address units (TA units) and 8 texture filtering units (TF units). TA units are completely decoupled from the SP, meaning that it's possible access them while SPs perform other operations, even using different clock frequencies. Embedded buffers store the output of the SP and can be quickly read by another SP for subsequent processing. This allows SIMD instructions to be "sistolically" implemented across clusters of stream processors.

In order to better exploit this device, we have adopted the technology CUDA™ (compute unified device architecture) [11] based on the C programming language. When programmed through CUDA, the GPU is viewed as a computing device capable of executing a very high number of threads in parallel. It operates as a coprocessor to the main CPU: a portion of an application that is executed many times, but independently on different data, can be isolated into a function that is executed on the device as many different threads.

Moreover, because of their algorithmic nature, we have optimized the kernels computing SVD and MLNPC including CUBLAS library [12], an efficient implementation of BLAS (Basic Linear Algebra Subprograms), which allows access to the computational resources of NVIDIA GPUs.

The modules computing SVD, Haar/Daubechies DWTs, and MLNPC have been implemented in custom ".cu" CUDA files. These files, as evidenced by a scheme in Fig. 4.c and an exemplificative code shown in Fig. 5, consist of a part based on custom CUDA kernels (labels KC), and another part based on CUBLAS library (labels CL).

Any call to a CUDA kernel must specify the execution configuration for that call. This configuration appears in the form <<<dimGrid, dimBlock[, numBytes] [, associatedStream]>>> between the function name and the parenthesized argument list, where dimBlock and dimGrid specify how the resulting data stream is partitioned (i.e., into a grid composed by dimGrid blocks, each one having dimBlock elements, representing the threads), numBytes and associatedStream are optional arguments, respectively specifying the bytes of shared memory that the call dynamically allocates per each block in addition to the statically allocated memory, and the associated stream (both of them default 0). For instance, the call "ReplicateCol<<<401,12, 200>>>(...)" partitions its computing into a grid of 401 blocks, each one having 12 threads and 200 bytes of additional dynamic memory; while the lines "dim3 block(4,4);", "dim3 grid(1000,6);" and "HAAR2D <<<grid,block>>>(...);" configure the call to HAAR2D with a grid of 1000x6 blocks, each one mapped in 4x4 threads. Anyhow, CUDA standardizes the maximum number of threads per block (dimBlock≤512) and of blocks (dimGrid≤65,535).

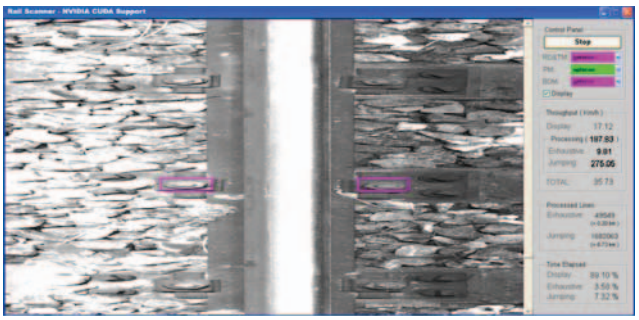

Fig. 6. Monitor. At the moment of this snapshot, the detection of left and right bolts is signaled.

Processed lines	3,032,432 [lines]	9.097 [km]
Elapsed time (e.t.)	175.37 [sec]	
Throughput (t.p.)	17,292 [lines/sec]	186.75[km/h]
Jumped lines	2,980,012 [lines]	98.27 %
Jumping search e.t.	117.70 [sec]	67.12 %
Jumping search t.p.	25,319[lines/sec]	273.45[km/h]
Exhaustive lines	52,420 [lines]	1.73 %
Exhaustive search e.t.	57.67 [sec]	32.88 %
Exhaustive search t.p.	909 [lines/sec]	9.82 [km/h]

Table 1. Obtained Performance.

The produced ".cu" files have been translated by the nvcc CUDA compiler [13] using the NVIDIA script NVMEX [14], a command which translates CUDA files into executable files (".mexw64" or ".mexw32") that can be directly called from MatLab, as if they were MATLAB built-in functions.

We have chosen MatLab as target environment for our system not only because it allocates matrices in memory coherently as CUBLAS does (i.e., by columns, contrarily as C compilers that works by rows), but also for exploiting the debugging and the fast prototyping facilities, as well as the efficient tools for creating graphic interfaces provided by MatLab (see Fig. 6).

5. ACCURACY AND COMPUTING PERFORMANCE

Accuracy in detecting the presence/absence of bolts was measured over a sequence covering more than 9 kilometers of railway and containing 3,350 hexagonal bolts. The system detected 99.9% of the visible bolts, 0.1% of the partially occluded bolts and 95% of the absences (Table 1).

The recognition rate resulted even more accurate using as benchmark a sequence of more than 6 kilometers containing 3,200 hook bolts: in this case, the detection of both absent and present elements was 100%, getting also an acceptable detection rate more affordable than in case of partially occluded hexagonal bolts (47% and 31% respectively for left and right), which makes the system. This higher accuracy is justified since the hexagonal shape causes more frequent miss classifications because of their similarity with the stones on the background.

Thanks to its parallel implementation, the obtained throughput (measured disabling the display) resulted 187 km/h. This one, compared to the throughput obtained on a Dual Quad-Core AMD Opteron™ Processor 2352 (8 CPUs) clocked at 2.11 GHz with 8 GB of RAM, represents a speed-up of 287 %, meaning a 23x speedup with respect to a single CPU.

6. CONCLUSION AND FUTURE WORK

This paper has proposed a visual system able to detect the bolts that secure the rail to the sleepers.

The implemented prediction module and the GPU-based parallel implementation allow to speed up the system performance in terms of the inspection velocity: the system analyses video at 273 km/h (Jumping phase) and at 10 km/h (Exhaustive phase), reaching a composite throughput of 187 km/h, for the considered benchmark. If the system remains in the Jumping phase for longer time, performance can increase subsequently. Next work can 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 phase is executed in continuous).

Obviously, the easy portability of the system onto more powerful computing solutions (e.g., the teraflop many-core processor NVIDIA Tesla provided with 960 cores per processor) can assure even higher throughput.

However, also considering the current performance, the proposed system constitutes a significant aid to the railway safety issue, since its high reliability, robustness, accuracy and computing performance allow a more frequent maintenance of the entire railway network.

7. REFERENCES

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

[2] K Sato, H. Arai, T. Shimuzu, and M. Takada, "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.

[3] W. Xishi, N. Bin, and C. Yinhang, "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.

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

[5] www.matrox.com/imaging/products/odyssey_xcl/home.cfm

[6] www.coreco.com

[7] "CAMERALINK: specification for camera link interface standard for digital cameras and frame grabbers," www.machinevisiononline.org

[8] Daubechies I. "Orthonormal bases of compactly supported wavelets," *Comm. Pure & Appl. Math*., vol. 41, pp. 909-996. 1988.

[9] Strang G., & Nuguyen T., *Wavelet and Filter banks*, Wellesley College, 1996.

[10] NVIDIA GeForce 8800 GPU Architecture Overview, NVIDIA Technical Brief, November 2006.

[11] NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 1.1, 6/23/2007.

[12] NVIDIA CUDA, CUBLAS Library, PG-00000-002_V1.0, June, 2007.

[13] NVIDIA CUDA, The CUDA Compiler Driver NVCC, 11/5/2007.

[14] NVIDIA, White Paper, Accelerating MATLAB with CUDA™ Using MEX Files, September 2007.