Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 1 (2009) 163-174

The Mojette Transform Tool and Its Feasibility

Péter SZOBOSZLAI[1], Jan TURÁN[2],
József VÁSÁRHELYI[3], Péter SERFŐZŐ[4]

[1] Magyar Telekom, Budapest, Hungary,
e-mail: szoboszlai.peter@telekom.hu
[2] Department of Electronics and Multimedia Communications,
Technical University of Kosice, Košice, Slovak Republic,
e-mail: jan.turan@tuke.sk
[3] Department of Automation, University of Miskolc, Miskolc-Egyetemváros, Hungary,
e-mail: vajo@mazsola.iit.uni-miskolc.hu
[4] Ericsson Hungary Ltd, Budapest, Hungary, e-mail: peter.serf.z@ericsson.com

Manuscript received March 15, 2009; revised June 10, 2009.

Abstract: The Mojette Transformation Tool (MTTool) is an implementation of the Direct Mojette transform and its inverse in Net environment. In contrast with the hardware development (MoTIMoT) [1], the software development provides us both an endless possibility of different variations of the Mojette Transform in a shorter time frame and lower costs. Tests with such a tool are much easier and it is also better for demonstration and training purposes. This paper tries to outline how the MTTool could be helpful for further developments both in software and hardware development.

Keywords: Mojette Transform, MoTIMoT, MTTool, performance test, image processing, software development.

1. Introduction

The Mojette Transform (MT) originates from France where J-P. Guédon referred to an old French class of white beans, which were used to teach children computing basics of arithmetic with simple addition and subtraction. He named it after the analogy of beans and bins. Bins contain the sum of pixel values of the respective projection line [2]. There are several different variations of MT applications nowadays which are used in different areas, such as tomography [3], internet distributed data bases [4], encoding, multimedia error correction [5], or The Mojette Transform Tool (MTTool), which was created for testing purposes. Moreover, it can be used for demonstrations and training purposes as well.

Although the MTTool development has not been finished yet, we have already gained much experience with it, and we can see how it may become more helpful for further projects both in software and hardware development. So the main purpose to build such an environment is that with its help we could try to compare MT software version with the hardware one. Possible application of the SW and the HW can be a surveillance system, where the captured and transformed data is stored on different servers for security reasons. From one transformed data the recorded data cannot be restored and loosing connection to one storage server is not affecting the restoration of the requested data.

2. Mojette and Inverse Mojette Transform

Mojette Transform: The main idea behind the Mojette transformation (similarly to the Radon transformation) is to calculate a group of projections on an image block [6]. The Mojette transform (MOT) (see [7], [8] and [9]) projects the original digital 2D image:

$$F = \left\{ F\left(i, j\right); i = 1, ..., N; j = 1, ..., M \right\} \tag{1}$$

onto a set of K discrete 1D projections with:

$$M = \left\{ M_k\left(1\right); k = 1, ..., K; 1 = 1, ..., 1_K \right\}. \tag{2}$$

MOT is an exact discrete Radon transform defined for a set $S = \{(p_k, q_k), k = 1, ..., K\}$ specific projections angles:

$$M_K\left(l\right) = proj\left(p_k, q_k, b_l\right) = \sum_{(i,j)\in L} F\left(i, j\right)\delta\left(b_l - iq_k - jp_k\right), \tag{3}$$

where $proj\ (p_k, q_k, b_l)$ defines the projection lines p_k, q_k, $\delta(x)$ is the Dirac delta with the form:

$$\delta\left(x\right) = \begin{cases} 1, if\_x = 0 \\ 0, if\_x = 1 \end{cases} \tag{4}$$

and

$$L = \left\{ (i, j); b_l - iq_k - jp_k = 0 \right\} \tag{5}$$

is a digital bin in the direction θ_k and on set b_l.

So the projection operator sums up all pixels values whose centers are intersected by the discrete projection line l. The restriction of angle θ_k leads both

to a different sampling and a different number of bins in each projection (p_k, q_k). For a projection defined by θ_i, the number of bins n_i can be calculated by:

$$n_i = \left(N-1\right)\left|p_i\right| + \left(M-1\right)\left|q_i\right| + 1 \tag{6}$$

The direct MOT is depicted in *Figure 1* for a 4x4 pixel image. The set of three directions *S={(-1,2),(1,1),(0,-1)}* results in 20 bins.

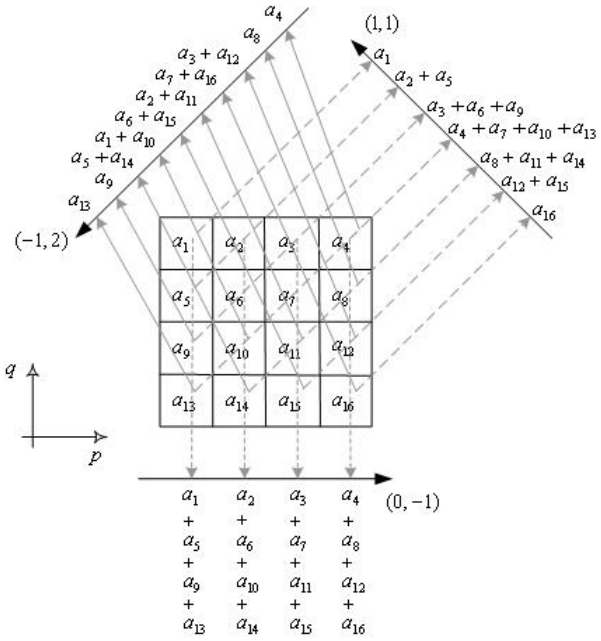

Figure 1: The set of three projections computed from a 4x4 image.

The MT can be performed by direct addition of the image pixel values in grey scale images, and for bitmaps we can add the different bitmap color table values.

Inverse Mojette Transform: The basic principle of the inverse Mojette transform is the following. We start the image reconstruction with bins corresponding to a single pixel summation. This reconstructed pixel value is then subtracted from the other projections and the process is iterated for the N^2-*1* pixels: the image is then completely decoded. In the case of a 4x4 pixel image reconstruction, if the directions of the MT sets are *S={(-1,2),(1,1),(0,-1)}*, then the minimum number of subtractions needed is 10, from the 20 bins. So should

it happen to lose some of the bins we could still reconstruct the image due to the redundancy of the MT.

3. Mojette Transform in MTTool

In MTTool the implementation of the MT was applied in three different ways. This is due to the fact that this application is still under development and the three different ways were constructed not at the same time, but in the previous years.

Table 1: MT implementation and its main differences

Nr.	Image Format	Projections	MT and Inverse MT
1	PGM	p={1,-1,3,-3}; q={quarter of the image size}	addition and subtraction
2	BMP	p={2,-2}; q={1} and p={3,-3,2}; q={1}	addition and subtraction
3	BMP	p={2,-2}; q={1} and p={3,-3,2}; q={1}	Matrix

The First Version: In the initial release one of the hardest decisions was to declare some rules, which had to be both flexible and at the same time not very complex. We had to declare the image sizes we had to work later with, and to look for a useful relationship between the picture size and the vectors we use in the MT, Inverse Mojette Transform (IMT). Considering several different file sizes, it was clear the smallest image size which can be used in real system is the 256 x 256 so, we decided to take the picture size $2^n x 2^n$, where n is equal to 8 and 9, but can be changed easily later on. So the transformable picture size are 256 x 256 and 512 x 512. In the Picture Preview we can open and display any kind of PGM or BMP file irrespective of the picture size, but some of the images are increased or decreased to fit on the screen.

Table 2: Image display in Picture Preview

Original size	Displayed size	Ratio
1600 x 1200	400 x 300	0,25
1599 x 1199	799 x 599	0,5
1024 x 768	512 x 384	0,5
Height < 1024	Height +180	Other

After checking the restrictions, the first step in the MT is to make a vector from the pixels of the image. When following a simple rule (*1*, 2^n *x* 2^n), it is easy to define the size of this vector. If *n=8,* this result in the vector (*1, 65536*), in which every line contains a pixel value from the picture. Because the PGM picture is a 256 greyscale image, a PGM file contains pixel values only from 0 to 255. In case of a BMP image, we could make it three times because of the different bitmap color table values.

In the second step we make the Mojette Transformation. The vector *p* is predefined for the four projection directions and the *q* vector has the same value in each case (quarter size of the 2^n *x* 2^n image). We generate four files for the four different projections, which are the following:

- originalfilename.pgm.moj1 (*1, q*)
- originalfilename.pgm.moj2 (*-1, q*)
- originalfilename.pgm.moj3 (*3, q*)
- originalfilename.pgm.moj4 (*-3, q*).

From the existing MT files (moj1, … moj4), we get the original PGM picture with the IMT. In this case all of the four Mojette Transformed files are needed to rebuild the original image without any errors at all. If any of the Mojette Transform files is defect or incomplete, the Inverse Mojette Transform will not give back the original image. Each of the four files contains a vector described above. The next step of the IMT is to read the first and last vectors of the third and fourth MT files and put them in their place. So we have in all four corners of the picture the valid pixel values filled up. See step 1, 2, 3 and 4 on the following figure:

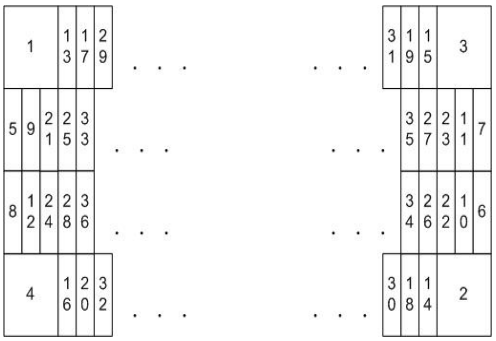

Figure 2: First 30 steps of the IMT.

After recreating the pixel values, we only need to add the new header for the file and the restoration of the original image is already performed.

The Second and Third Version: These solutions differ from the previous one in such a way that these are applied on BMP images and in these cases we perform the MT and IMT on the three different bitmap color tables. We use the same algorithm for the three different color maps and collecting the bins into 3 separate files which differ in their extensions and of course in their content. On the bitmap images we use the directions $S_1=\{(2,1),(-2,1)\}$ and $S_2=\{(3,1),(-3,1),(2,1)\}$ for the block sizes 4 and 8. Although the MT is also prepared for the block size 16 and 32, the implementation of the IMT isn't done yet. In the second version, we use simple addition and subtraction – different from the one mentioned in the first version –, since here we have block sizes 4 and 8 and there we perform the MT and IMT on the whole image at once and not step by step. In the third version, instead of addition and subtraction, we use matrices for the MT and IMT on the above mentioned block sizes. The MT with matrices is implemented in the following way, where b_i is the bin resulted from the following equation:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix} = \begin{bmatrix} 10 \\ 123 \\ 37 \\ 137 \\ 254 \\ 319 \\ 433 \\ 68 \\ 6 \\ 234 \\ 125 \\ 267 \\ 312 \\ 8 \\ 45 \\ 178 \end{bmatrix} \quad (6)$$

The inverse matrix for the previous example (for the 4x4 matrix size) is implemented as it is shown in the next equation, where a_i stands for the original values of the matrix:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \end{bmatrix} = \begin{bmatrix} 10 \\ 123 \\ 25 \\ 35 \\ 12 \\ 102 \\ 252 \\ 241 \\ 2 \\ 78 \\ 255 \\ 23 \\ 178 \\ 45 \\ 6 \\ 234 \end{bmatrix} \quad (7)$$

4. MoTIMoT implementation

The ‘MoTIMoT’ co-processor denotes the hardware implementation of the direct MT and IMT as co-processing elements of an embedded processor. The advantage in using FPGAs for this is the flexibility of the development tools, the hardware-software co-design solutions, and the compact hardware in the loop simulation. The embedded reconfigurable hardware is based on Xilinx ML310 board [10].

Starting from a 256x256 pixel size greyscale image, this requires 64KB memory. In order to process all the projection lines in parallel (in the case of the MT), one needs as many 64KB sized memory blocks (i.e. Block RAM) as many projection lines we have for this image size. The bin vectors are stored in the external memory in a so-called ‘MT memory file’. The division in slices of the original image is motivated by the fact that this can be corrupted during the transmission of the MT file. The image can not be reconstructed without the damaged area from the corrupted MT file. While applying the MT to slices, the effect of the MT corruption is diminished. Similarly, the memory necessary to calculate the MT and IMT is smaller in the case of slices. The whole image process would result in a need for reconfiguration in order to calculate all the projections, because 256KB are needed to load the 256x256 image in the embedded memory only for 4 projection lines (the XC2VP30 has 1.7Mb Block RAM ≈ 212KB). For this reason the 256x256 pixel image is divided in 4 slices (128x128).

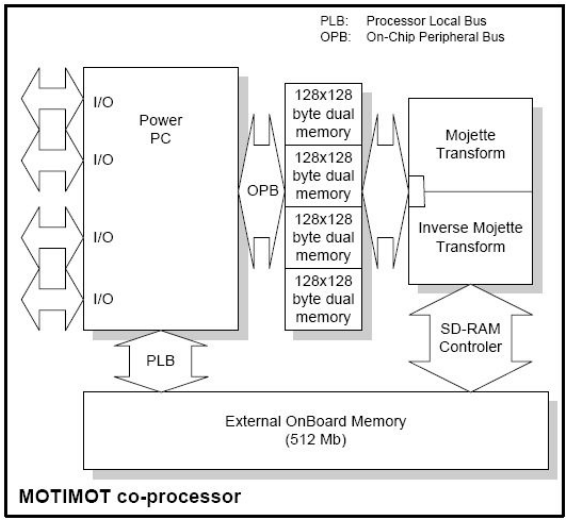

Figure 3: MoTIMoT co-Processor Block Scheme [1].

The main parts of the image processing systems are: the PowerPC as the main processing unit, the MoT unit and the IMoT unit. Both MoT and IMoT processors are connected to the PowerPC via the internal PLB bus, because their work runs under the main processor control.

While processing the IMT (for the same image size and projection lines) one needs to read the MT memory file from the external memory and to reconstruct the image. The bin size means the maximum pixel numbers contained by a bin and also defines the number of bits needed for the unary bins of the unary MT file. Bins containing only one pixel value are placed to their corresponding position during the back projection (IMT). These pixel values contained by other bins (in other projections) as well and which bin values have to be decreased with the current pixel value. To calculate the positions of the bins in the projections and to calculate the correspondence in the image of a single pixel bin we need the unary image. Thus when the value of a single pixel bin is substituted and the other bin values are decreased, the changes have to be validated as well in the unary MT file. The unary MT file contains unary bins. These bins contain not only the current pixel number included in the bin in the MT file, but the corresponding position in the image of these pixels too.

5. Experiments and results with MTTool and MoTIMoT

MTTool: We can decrease the size of any vectors which are created from the projections of MT with the built in ZIP and Huffman coding opportunities. The Huffman lossless encoding and decoding algorithm was chosen due to its binary block encoding attribute and not because of its compression capability. Good data compression can be achieved with Zip and Unzip, which are also implemented. The possibility of time measuring with simple tools, such as labels or easily generated text files which include the test results, can give us a good insight into the MT and IMT. From these results we can estimate and predict the consumed time on hardware implementation and its cost as well.

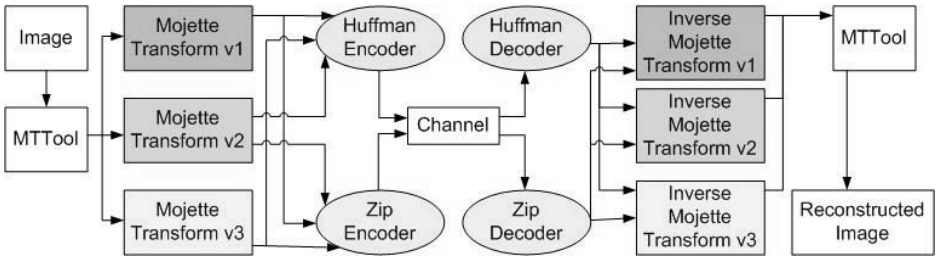

Figure 4: Logical system architecture of the MTTool.

The time measurement was applied on three different images with three different image sizes and with three different periods. The images were black and white PGM files with pixel values of 0 and 255 and the LENA.PGM. The first test ran only once, after which the second test ran for 6 times in a row, and the last test ran 24 times. Each test was performed with sizes of 16x16, 32x32 and 512x512. The results of the two smallest image sizes are nearly identical, and the results were nearly always under 20 milliseconds for MT and IMT, but we could see the following difference regarding the 512x512 image size:

Table 3: Test result of the MT and IMT with the first version

IMAGE	Black (512x512)		White (512x512)		Lena (512x512)	
	Minute: Second: Millisecond	MT and IMT in Millisecond	Minute: Second: Millisecond	MT and IMT in Millisecond	Minute: Second: Millisecond	MT and IMT in Millisecond
MT start	57:14:277		3:45:510		21:36:79	
MT end IMT start	57:15:439	1162	3:47:403	1893	21:37:762	1683
IMT end	57:15:910	471	3:47:964	561	21:38:303	541
MT start	57:22:259		4:0:822		21:49:749	
MT end IMT start	57:23:411	1152	4:2:555	1733	21:51:391	1642
IMT end	57:23:891	480	4:3:105	550	21:51:932	541

From this table we can see that the difference between black and white images is more than 50 percent, when it comes to the MT, and only 20 percent when we apply the IMT on the Mojette files. For a real time video surveillance application which should capture at least 25 image per second (PAL) this result is not enough.

MOTIMOT: The simulations were made on PC hardware environment using a portable greymap 256x256 image (Lena) without transmission and no bit-corrupted errors, just as in the MTTool. The simulation proved the correctness of the implemented algorithms and the functionality of the proposed hardware. In both implementation of the MT and IMT the images were restored with only small differences. The original creation date of the image is replaced with the date when the IMT was performed. All the header information is cut and isn't restored later on. Restoration of the image includes only the pixel values of the original image. We also attach a new automatically generated header to these pixels, so the restored image pixel values are exactly the same as the pixel values in the original image. Therefore any information included in the image itself such as watermark, time and date etc. can be restored later on easily.

6. Conclusion

The paper outlines the different ways, how the Mojette Transform is currently implemented in the MTTool and also gives an insight how the hardware implementation of Mojette and inverse transformation in the embedded system using FPGA has been done. The original contribution is the calculation of the dimension of Mojette memory file, the definition and analysis of the hardware structure as a whole with the simulation results. Future work is needed both in the software and hardware versions. In the software version (MTTool) more tests should be performed to get more accurate results, and by comparing them to the results of the hardware, we should find an optimal way to perform the Mojette Transform. In the hardware version (MoTIMoT), finalizing the implementation of the co-processors as a whole with run-time reconfiguration is needed.

Acknowledgements

The authors gratefully acknowledge the donations of Xilinx Inc. and Celoxica Inc., which made it possible to start this research.

Thanks for Ferenc Nagy who offered the necessary space and time for our work.

References

[1] Serfőző, P., Vásárhelyi, J., "Development work of a Mojette transform based hardware codec for distributed database systems", in *Proceedings of 8th Interational Caroathian Control Conference ICCC2007, Strebske Pleso, Slovakia,* 2007 May 24-27, pp. 631-635.

[2] Guédon, J.-P., Normand, N., "The Mojette transform: The first ten years", in *Proceedings of DGCI 2005*, LNCS 3429, 2005, pp. 79-91.

[3] Guédon, J.-P., Normand, N., "Spline Mojette transform application in tomography and communication", in *EUSIPCO*, Sep. 2002.

[4] Guédon, J.-P., Parrein, B., Normand, N., "Internet distributed image databases", *Int. Comp. Aided Eng.*, Vol. 8, pp. 205–214, 2001.

[5] Parrein, B., Normand, N., Guédon, J.-P., "Multimedia forward error correcting codes for wireless LAN", *Annals of Telecommunications (3-4),* pp. 448-463, March-April, 2003.

[6] Normand, N., Guédon, J.-P., "La transformee Mojette: une representation recordante pour l'image", *Comptes Rendus Academie des Sciences de Paris, Theoretical Comp. SCI. Section*, 1998, pp. 124–127.

[7] Katz, M., "Questions of uniqueness and resolution in reconstruction from projections", Springer Verlag, Berlin, 1977.

[8] Autrusseau, F., Guédon, J.-P., "Image watermarking for copyright protection and data hiding via the Mojette transform", in *Proceedings of SPIE*, Vol. 4675, 2002, pp. 378–386.
[9] Turán, J., Ovsenik, L., Benca, M., Turán, J. Jr., "Implementation of CT and IHT processors for invariant object recognition system", *Radioengineering*, Vol. 13, No. 4, pp. 65-71, Dec. 2004.
[10] Xilinx, ML310 User Guide, pp. 73, http://xilinx.com.