

# IMAGE RECONSTRUCTION AFTER A SUCCESSION OF 2:1 DOWNSAMPLING – UPSAMPLING

Emil Calofir<sup>1</sup>  
Radu Ionut Dan<sup>2</sup>  
Vlad Lionte<sup>3</sup>  
Ion Bucur<sup>4</sup>

## ABSTRACT

*Video streaming had a long way to walk from its first days in the 90's until nowadays. In the year 2012 people were watching over 4 billion video clips every day on YouTube, according to Reuters. The transmission of video information towards unstable networks becomes a problem and new methods are created in order to prevent this kind of situations. In this work is presented such a method, which confers to some user with no stable internet connection a nice and continuous streaming experience.*

**KEYWORDS:** video streaming, network, compression

## 1. INTRODUCTION

This project has the purpose to streamline the transmission of images in a stable network, so that at the receiver these should recombine having a quality as high as possible.

The image data package transmission was realized by implementing the upsampling and downsampling transformation over the images.

This work is composed from two subsystems: client and server. Between these two we can create any type of connection; the tests are performed in local module with an adjustable latency level. Therefore, we are testing the disruptive factors that generate noises and delays in the process of sending data packages.

In the first part of the work we present the main projects that we have studied in this domain.

The second part contains information about what we have purposed, the method we wanted to develop, the description of our system and the implementation of the algorithms.

In the final part of the study, we present the experimental results, evaluation of the tests regarding the performance and the quality.

## 2. THE CURRENT STAGE

Downsampling of compressed images in the discrete domain can be observed in [1]. The technique is composed by two steps: first the image is passed through an antialiasing filter, and then is sub-sampled with an arbitrary factor for every dimension. In the case of discrete compressed images, they should be brought in the spatial domain by reversing the DCT

---

<sup>1</sup> Engineer, emil.calofir@aut.pub.ro, "Politehnica" University of Bucharest, 060042 Bucharest, Romania

<sup>2</sup> Engineer, radu.dan0210@aut.pub.ro, "Politehnica" University of Bucharest, 060042 Bucharest, Romania

<sup>3</sup> Engineer, vlad.lionte@cti.pub.ro, "Politehnica" University of Bucharest, 060042 Bucharest, Romania

<sup>4</sup> Associate Professor PhD Eng., ion.bucur@cs.pub.ro, "Politehnica" University of Bucharest, 060042 Bucharest, Romania

(discrete cosine transformation) and afterwards to be passed through the filtering procedures.

A direct approach would stand in working in the compression domain, where both operations (filtering and downsampling) are combined in the DCT domain. This can be realized by cutting down the DCT coefficients for higher frequencies and using the reversed DCT with a significantly smaller number of coefficients in order to reconstruct the low resolution image. For example, one can use 4x4 coefficients from 8x8, and apply the reversed DCT over these coefficients in order to reduce the resolution two times for every dimension. It has been noticed that this approach, also named frequency masking, does not lead to significant improvement of the compression and it needs the encoders and decoders to be able to accept 4x4 DCT and 8x8 reversed DCT. It also needs that the coding schemes to be optimized for the case 4x4. More than this, the method may induce blockages and distortions because of the bad approximation that occur after discharging high-order coefficients. [2]

Experimental results from [1] can be seen in Table 1, and the used methods can be observed in fig. 1.

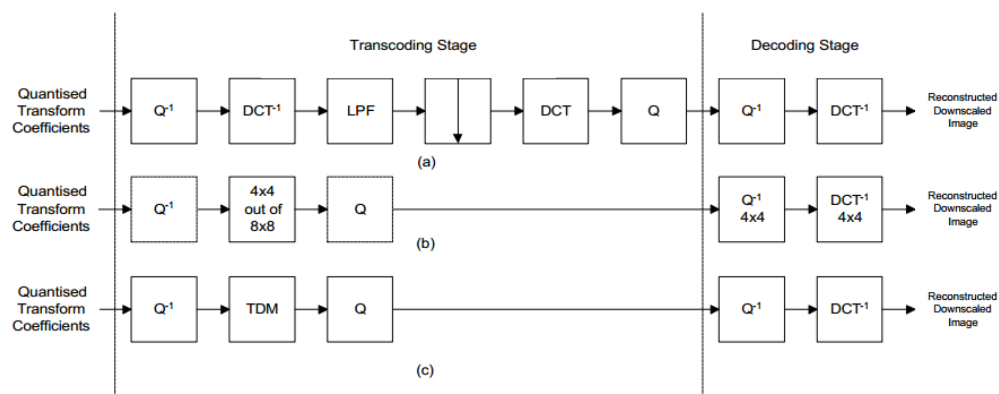


Figure 1. Application design

| Image                 | Method (a)          | Method (b)          | Method (c)          |
|-----------------------|---------------------|---------------------|---------------------|
| Lenna<br>(256x256)    | 23.3 dB<br>8.8 KB   | 32.42 dB<br>36.7 KB | 27.14 dB<br>12 KB   |
| peppers<br>(256x256)  | 23.57 dB<br>9.2 KB  | 31.97 dB<br>36.3 KB | 27.58 dB<br>12 KB   |
| sailboat<br>(256x256) | 21.59 dB<br>11.2 KB | 31.56 dB<br>42.2 KB | 25.54 dB<br>15.7 KB |
| foreman<br>(88x72)    | 22.89 dB<br>1.3 KB  | 32.82 dB<br>4.3 KB  | 26.67 dB<br>1.7 KB  |

|                     |                     |                     |                     |
|---------------------|---------------------|---------------------|---------------------|
| news<br>(88x72)     | 16.18 dB<br>1.2 KB  | 27.29 dB<br>3.9 KB  | 20.87 dB<br>1.7 KB  |
| target<br>(256x256) | 18.57 dB<br>8.5 KB  | 23.21 dB<br>33.6 KB | 23.96 dB<br>12.2 KB |
| hotel<br>(360x288)  | 19.57 dB<br>16.6 KB | 29.85 dB<br>60.6 KB | 24.02 dB<br>23.1 KB |

Table 1. Experimental results of down-sampling compressed images [1]

In [3][10][11] is presented an unique system of image reconstruction, where every component has a package – a unique set of upsampling – downsampling filters – which can be used in order to recursively reconstruct a pyramid. The construction algorithm of the pyramid is based on successive downsampling operations, followed by upsampling and residues encoding operations.

### 3. SYSTEM IMPLEMENTATION

The main reason in using a downsampling operation is to reduce the data quantity. Assuming that we have an audio signal with a frequency rate of 44,1 kHz, and the processing algorithm of audio signals can only use signals at 22,05 kHz. Therefore, we will reduce the number of samples from the initial signal. We can apply this principle for any type of signal. Only because we have more information it is not binding to need all of it.

#### Downsampling operators

##### *Decanting*

Assures the aliasing effect removal;

Can be sequentially applied over all axes.

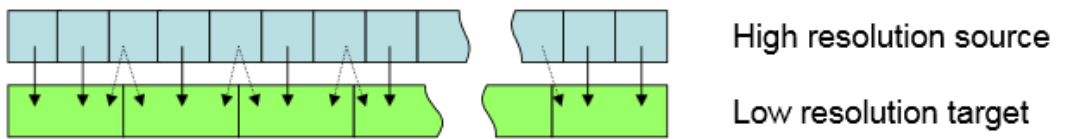


Figure 2. Decanting downsampling

##### *Bilinear interpolation*

The calculation of the average intensity for the nearby pixels:  $a_j = (1-\alpha)I_{i,j} + \alpha I_{i+1,j}$

$a_{j+1} = (1-\alpha)I_{i,j+1} + \alpha I_{i+1,j+1}$ , calculation of the average value on the x axis

$I_{k,l} = (1-\beta)a_j + \beta a_{j+1}$ , calculation of the average value on y axis

Needs extra finishing operation in order to remove the aliasing effect

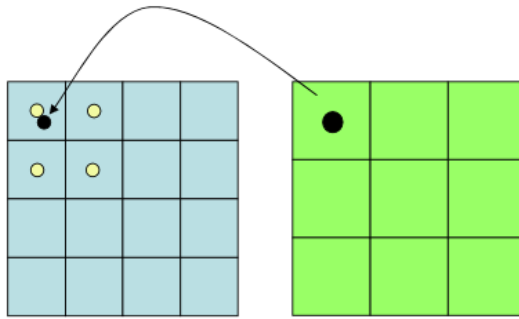


Figure 3. Bilinear downsampling operation

By performing an unsampling operation, we usually try to generate extra information. for example: for an image are generated intermediate pixels. Of course, another reason is to mix two signals that were sampled at different frequencies.

### Upsampling operators

#### *Decanting*

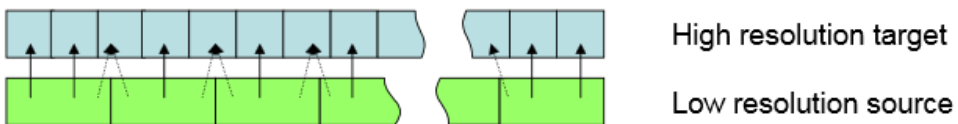


Figure 4. Decanting upsampling

#### *Bilinear interpolation*

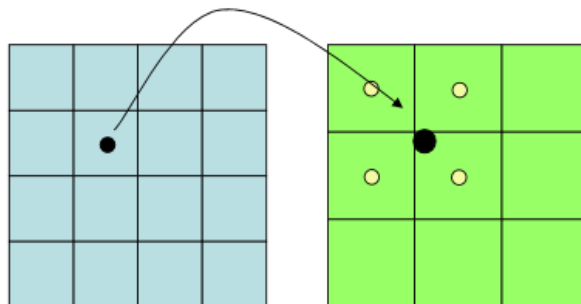


Figure 5. Unsampling bilinear

In the implementation presented in this study, we will use the bilinear interpolation operator for downsampling and upsampling.

This project was implemented using Microsoft Visual Studio 2010, language Visual C#. For image manipulation we integrated the library EmuCV5.

Graphic interface with user for server subsystem is presented in fig. 6:

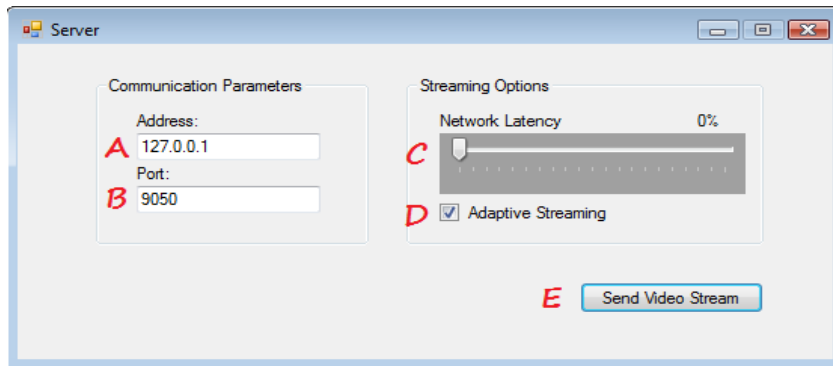


Figure 6. Graphic interface with user for server subsystem

Where:

- A: textbox where the user introduces his server IP (localhost for default)
- B: textbox for introducing the opened port for receiving the packages from the client
- C: network latency simulator, with an increment step of 2,5%
- D: flag Boolean for the adaptive data streaming, where true = ON, false = OFF
- E: sends to the client the processed video stream picked up from the web cam

When pressing the “Send Video Stream” button, a thread is opened for the socket connection (the IP is picked up from A, and the port from B).

The capturing of every frame picked up from the web camera is made through the method EmguCV capture(). While the thread is active, the downsampling and upsampling iterations of every image are executed.

### ***Iteration 1***

The original image is the capture from webcam. The image is encoded as bitmap through the function ToBitmap(). The downsampled image has half the size of the original image ( $\text{original.Width} / 2$ ,  $\text{original.Height} / 2$ ), and the unsampled bitmap image is composed from the downsampled image with the size of the original image ( $\text{downsampled.Width} * 2$ , respectiv  $\text{downsampled.Height} * 2$ ). Finally, the image with the details of iteration can be composed using the imgSub(original, upsampled) method.

### ***Iteration 2***

The original image has half the size of the webcam capture. Same as the Iteration 1, it is encoded as bitmap. The downsampled image has half the size of the original image ( $\text{original.Width} / 2$ , respectiv  $\text{original.Height} / 2$ , meaning that the image is 4 times smaller than the webcam image). The upsampled image is composed exactly like in the first iteraton, and the second iteration's image details can be composed through the imgSub method.

### *Iteration 3*

Works similar to the previous iteration, resulting the downsampled image 8 times smaller than the original image.

Finally, depending on the set network latency parameters and the adaptived streaming, detailed images are sent to the client in order to be processed later.

The end of the frame is marked by sending a 1x1 bitmap.

For a better quality of the details, the compression algorithm Huffman is applied.

The client subsystem is presented in fig. 7:

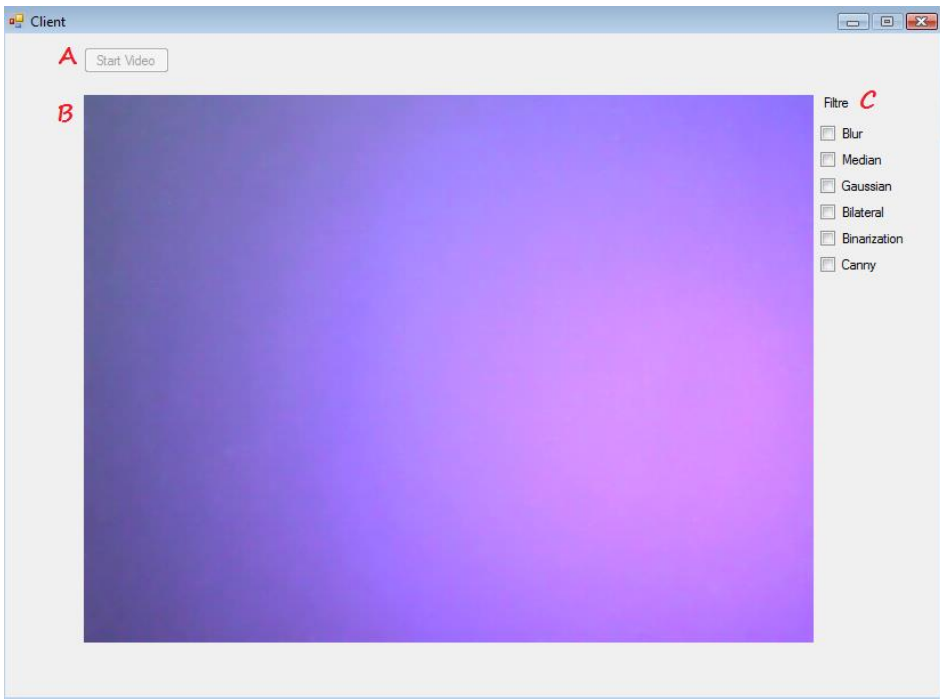


Figure 7. Graphic interface with the user for the client subsystem

Where:

- A: marks the start of the webcam images;
- B: resulted final image from the composition of the details received from the server, compressed through Huffman method;
- C: filters that user can apply to the image: blur, median, Gaussian, bilateral, image binary analysis and outline detection.

The action started by pressing the “start video” button opens the socket for listening the package received from the server.

A `MemoryStream` object is allocated, containing the octets sequences composing the images transmitted from the server. Further on, iterations 1-3 are interpreted, so that after the composing of a detailed image, we are testing if the end frame bitmap is received. If the images have been compressed at the server using the Huffman method, these are decoded before displaying.

The filters are applied to the detailed images, being stored as variable Boolean structure. When a user chooses a filter, real time verifications take place in the thread for listening the packages. For example, if smoothGaussian filter was selected, the detailed image received is taken through this filter and then it is displayed.

#### 4. EXPERIMENTAL RESULTS

For the experiments we used a A4Tech PK-760E webcam, with a sensor resolution of 0.3 MP, photo resolution of 5MP and a video resolution of 2560 x 2084, connected to the computer via USB. We used a desktop PC with an Intel Core 2 Duo E8400 3GHz, video card HIS ATI Radeon HD4870 512 RAM DDR5 256-bit and 4GB RAM DDR2. Tests were performed in the local network, simulating latency bigger and bigger.

First test was made for evaluating the performances of the frames sent per second (FPS), using the Huffman algorithm. The results are in the graphic from fig. 8:

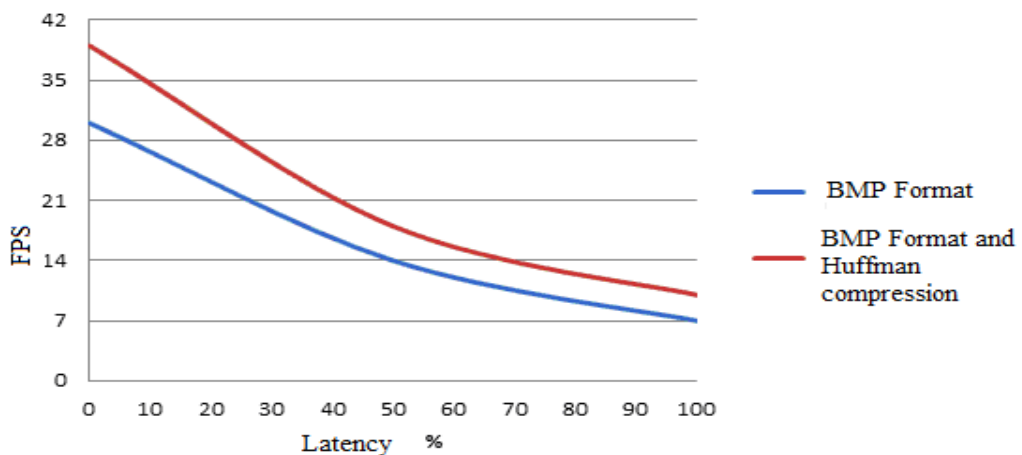


Figure 8. Performance in FPS terms with and without Huffman compression

We can notice that best performances we had took place when there is no latency in the network (0%), where we obtained an average value of 38 fps in sending bitmap image sequences with Huffman compression, or 29 fps without compression.

As a conclusion, we noticed that FPS increased by 30% when applying Huffman algorithm over detailed images.

The second test was performed by obtaining a subjective evaluation over the final image reconstruction performance at the receiver. We obtained the evaluation of a 5 persons group (3 boys, 2 girls) with ages between 24 and 26 years old, over the final image quality.

Every subject had to note between 1 and 10 the quality of 3 video sequences with the duration of 20 – 30 seconds, with disrupting factors from a lower level to a higher level. The results can be seen in table 2:

| <b>Subject</b>    | <b>Clip</b> | <b>Clip duration (seconds)</b> | <b>Disrupting level / camera movement</b> | <b>Subject note</b> |
|-------------------|-------------|--------------------------------|-------------------------------------------|---------------------|
| Subject #1 (boy)  | Clip #1     | 30                             | low                                       | 10                  |
|                   | Clip #2     | 25                             | average                                   | 10                  |
|                   | Clip #3     | 20                             | High                                      | 9                   |
| Subject #2 (girl) | Clip #1     | 30                             | Low                                       | 9                   |
|                   | Clip #2     | 25                             | Average                                   | 8                   |
|                   | Clip #3     | 20                             | high                                      | 8                   |
| Subject #3 (boy)  | Clip #1     | 30                             | Low                                       | 10                  |
|                   | Clip #2     | 25                             | Average                                   | 10                  |
|                   | Clip #3     | 20                             | high                                      | 9                   |
| Subject #4 (girl) | Clip #1     | 30                             | Low                                       | 10                  |
|                   | Clip #2     | 25                             | Average                                   | 10                  |
|                   | Clip #3     | 20                             | High                                      | 10                  |
| Subject #5 (boy)  | Clip #1     | 30                             | Low                                       | 9                   |
|                   | Clip #2     | 25                             | Average                                   | 8                   |
|                   | Clip #3     | 20                             | High                                      | 8                   |

Table 2. Final image quality evaluation at the receiver

Subjects watched video sequences, at first normal, without applying the upsampling – downsampling operations. This was necessary for them to get used to the webcam and to obtain a more relevant score for the algorithm evaluation. We notice a score of 9,2/10 when we have an average distortion level of the webcam and a score of 8,8/10 for intense movement of the camera.

The last test stands in applying the filters over the detailed images at the decoder. Its purpose was to obtain an increasing quality of the final image by applying an image filter. Best results were obtained with a Gaussian filter:

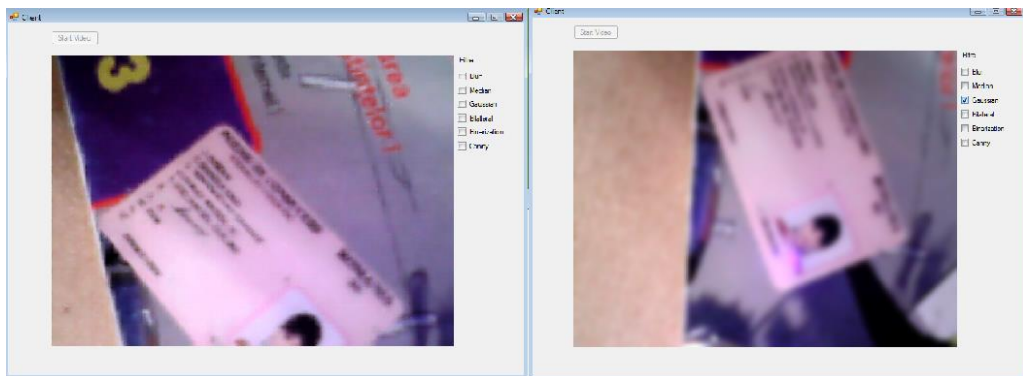


Figure 9. To the left: an image from the video sequence without a filter, to the right: capture with a Gaussian filter

## 5. CONCLUSIONS

We presented an implementation of downsampling – upsampling techniques which offer to any user an enjoyable and continuous experience in watching video sequences transmitted through an unstable network.

We also managed to streamline the calculation algorithms, obtaining a final image at the decoder with a very high quality.

The Huffman compression supports us in increasing the transmitting speed of images, therefore we managed to increase the fps number by 30%.

## 6. FUTURE WORK

In the future we intend to use Super-Resolution algorithms after applying the upsampling operation, in order to improve the quality of the image.

We also want to offer a variant of the system for mobile applications.

## REFERENCES

- [1] A. N. Skodras, C. A. Christopoulos, “Downsampling of compressed images in the DCT domain”, Electronics Laboratory, University of Patras, GR-26110 Patras, Greece; Computer Technology Institute, PO box 1122, GR-26110 Patras, Greece; Ericsson Telecom AB, Compression Lab, TN/ETX/PN/XML, S-126 25 Stockholm, Sweden.
- [2] Y. Noguchi, D. G. Messerschmitt and S.-F. Chang, “MPEG Video Compositing in the Compressed Domain”, Proc. IEEE ISCAS-96, 1996.
- [3] A. Enache, C. A. Boiangiu, "A Pyramidal Scheme of Residue Hypercubes for Adaptive Video Streaming", International Journal of Computers and Communications, Volume 8, 2014, pp. 128-133
- [4] C. W. Kok., “Fast Algorithm for Computing Discrete Cosine Transform”, IEEE Trans on Signal Processing, Vol. 45, No. 3, pp. 757-760, Mar. 1997.

- [5] V. Bhaskaran, K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", Second Edition, Kluwer Academic Publishers, 1997.
- [6] C. J. Hung, C. K. Huang, B. Y. Chen, "Image Super-Resolution by Vectorizing Edges", National Taiwan University, 1999.
- [7] C. A. Boiangiu, A. Olteanu, A. V.Stefanescu, D. Rosner, A. I. Egner, "Local Thresholding Image Binarization using Variable-Window Standard Deviation Response", Proceedings of the 21st International DAAAM Symposium, 20-23 October 2010, Zadar, Croatia, pp. 133-134.
- [8] T. Frajka, K. Zeger, "Downsampling dependent upsampling of images", University of California, San Diego, Department of Electrical and Computer Engineering, 9500 Gilman Dr, MC0407, La Jolla, CA 92093-0407, USA - 2 October 2003.
- [9] Daniel Lélis Baggio, Shervin Emami, David Millán Escrivá, Khvedchenia Ievgen, Naureen Mahmood, Jasonl Saragih, Roy Shilkrot, "Mastering OpenCV with Practical Computer Vision Projects" ISBN-10: 1849517827 - 3 December 2012.
- [10] Adrian Enache, Costin-Anton Boiangiu, "Adaptive Video Transmission Using Residue Octree Cubes.", Computers, Volume 3, Number 2, 2014, pp. 58-68
- [11] Adrian Enache, Costin-Anton Boiangiu, "Adaptive video streaming using residue hypercubes," in Proceedings of the 12th International Conference on Circuits, Systems, Electronics, Control & Signal Processing (CSECS '13), Budapest, Hungary, December 10-12, 2013, pp. 173-179.