Time multiplexing CNN simulator using RK7(5)

Osama H. Abdelwahed[1,2] and M. El-Sayed Wahed[2]
[1]Mathematics Department, Faculty of Science, Suez Canal University, Egypt
[2]Department of Computer Science, Faculty of Computers and Informatics, Suez Canal University, Egypt

Abstract

RK7(5) is a numerical method was used to minimize the local truncation error using selection step size algorithm. Time multiplexing CNN simulator was modified using RK7(5) to improve the performance of the simulator and the quality of the output image for edge detection. The results showed better performance than those in the literature.

Keywords —**Cellular neural network; Image processing; CNN simulator; RK7(5); Local truncation error**

1 Introduction

Edge detection is considered to be one of the essential steps in identifying an object in image processing. Many techniques have been proposed to detect edges in image processing using cellular neural network(CNN) [8]. CNN was introduced by Chua and Yang [3] in 1988. The main properties of CNN are their local interactions among their nearby cells in addition to their parallel processing which is considered to be one of the most important aspects in CNN. CNN has many applications in image processing like medical image segmentation, image denoising, and edge detection. Time multiplexing CNN simulator was proposed by C.C. Lee and Jose Pineda de Gyvez [6] in 1994. They used the numerical integration algorithms Euler, the improved Euler and Runge Kutta method to solve the differential equations which represent CNN. Their results for edge detection need to be improved and also the computation time need to be decreased. Another work presented by V. Murugesh and K. Murugesan[8]. They introduced the numerical methods, RK-Gill [10] and RK-Butcher [1, 2, 7], but unfortunately, the obtained results need to be improved. So we proposed RK7(5) [12] as a numerical integration method used to minimize the local truncation error using the selection step size algorithm [11]. Our proposed method showed better results than those in the literature. The obtained results included both faster performance and better edge detection.

RK7(5) is proposed to achieve two targets: First, to increase the efficiency of the time multiplexing CNN simulator by decreasing the needed simulation time; Second, to improve the quality of the edge detection results. The organization of the paper is as follows: section (2) presented CNN in details. In Section (3) time multiplexing CNN simulator was discussed. Section (4) presented the proposed method. Results of the performance of the simulator and edge detection were discussed in section (5). Section 6 discussed the Conclusion.

2 Overview of CNN

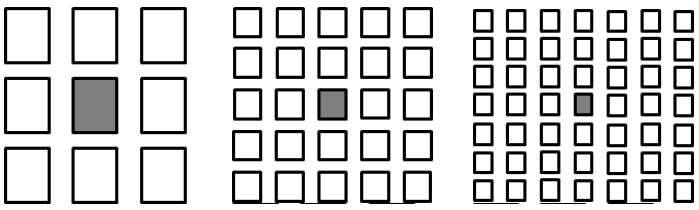
Cellular neural network is a method used both the concepts of neural networks and cellular automata in which its cells are connected locally through their neighbors and processed in parallel through their interactions [3].

The r-neighborhood of CNN is defined as in the following:

$$N_r(i,j) = \{C(k,l)\}\,|_{i=1,M;j=1,N}^{max}\,\{|k-i|\,,|l-j|\} \leq r, \tag{1}$$

Where $r > 0$.

The expression "1− neighborhood" is expressed for $r = 1$, " 2− neighborhood" is expressed for $r = 2$, "3− neighborhood" is expressed for $r = 3$, etc. as illustrated in Fig. (1).



(a) 1− neighborhood (b) 2− neighborhood (c) 3− neighborhood

Figure 1: Examples of $r-$ neighborhood

A two dimensional CNN with size equal to 4 has shown in Fig. (2), where each cell has interactions with its nearby cells locally for the 1− neighborhood case.

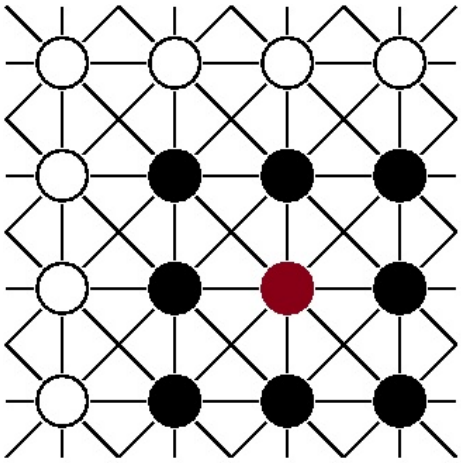


Figure 2: CNN structure with size = 4

The most important operators in CNN are the templates A and B which represent the output feedback and the input control respectively. They control the behavior of CNN in addition to the bias value I.

According to [3], the CNN equation is represented as in the following:

$$C\frac{dx_{ij}}{dt}(t) = -\frac{1}{R_x}x_{ij}(t) + \sum_{C(k,l)} A(i;j;k,l)y_{kl}(t) + \sum_{C(k,l)} B(i;j;k,l)u_{kl}(t) + I_{ij} \quad (2)$$

I_{yx} controls the behavior of the following output function y_{ij}:

$$f(x) = \frac{1}{2}\left(|x+1| - |x-1|\right) \quad (3)$$

Where,

$$I_{yx} = \frac{1}{R_y} f(x_{ij}) \quad (4)$$

The output function $y_{ij} = I_{yx}R_y$ can represented as in the following:

$$y_{ij} = f(x_{ij}(t)) = \frac{1}{2}\left(|x_{ij}(t)+1| - |x_{ij}(t)-1|\right) \quad (5)$$

The general form of CNN dynamics is represented by the following equation:

$$\frac{dx_{ij}}{dt}(t) = -x_{ij}(t) + \sum_{C(k,l)} A(i;j;k,l)y_{kl}(t) + \sum_{C(k,l)} B(i;j;k,l)u_{kl}(t) + I_{ij} \quad (6)$$

The different values of the templates A and B and the bias value I determines the type of application used in CNN. For example, edge detection, noise removal, or pattern recognition has different template values.

The templates of CNN are space invariant i.e. for example the template $B(i,j;i+1,j)$ is the same for all values at (i,j).

3 Time Multipexing CNN simulator

The time multiplexing CNN simulator [6] was proposed to solve the problem of hardware implementation. From hardware point of view, may be there are thousands of pixels need to be implemented using the single layer CNN simulator [5]. It is very difficult to manage this problem because of the limitation of the hardware. Hence the time multiplexing algorithm was proposed to fix this complex problem by dividing the input image into a number of sub-images whose size equal to to the size of the block of CNN processors. In this case, each block will process a sub-image using single layer CNN simulator till it converges. The algorithm continues for the next sub-images using the same procedure till the whole image will be processed [6]. Although the algorithm succeeded to solve the problem of the hardware limitation, some side effects appeared. The border pixels for each sub-image have incorrect results because there is no information for the neighbors which results in two erroneous values. The first error is occurred because of the effect of the template B. This error is computed using the following formula [9]:

$$E_{ij}^{B} = \sum_{i=1}^{3} b_{i,j+1} sign\left(u_{i,j+1}\right) \quad (7)$$

Where $b_{i,j+1}$ represents the values missed by using the template B due to the absence of the input signals $u_{i,j+1}$. And $sign()$ is the sign function which is used to represent the pixel status either black or white. This error depends on the input image and the template value.

The second error appeared as a result of the effect of the template A. This error is evaluated using the following equation:

$$E_{ij}^{A} = \sum_{i=1}^{3} a_{i,j+1} y_{i,j+1}(t) \quad (8)$$

Since $a_{i,j+1}$ is the error obtained due to the template A

because of the absence of the values of $y_{i,j+1}(t)$.

To minimize the errors E_{ij}^B and E_{ij}^A , we exploit the advantage of CNN in which there is a local interaction among neighboring cells. First, to minimize the error, E_{ij}^B , we chose a belt of pixels from the input image equals to the radius of the neighborhood of CNN around each sub-image. Second to minimize the error, E_{ij}^A, the pixels between adjacent sub-images were overlapped and each cell in the border can interact with its neighbors as interior cells. The minimum overlap width is twice of the neighborhood's radius of the CNN [9].

Although overlap property improved the interaction among neighboring cells, the simulation time increased. Fortunately, the big chance of having the pixels of each sub-image all black or all white helped in reducing the simulation time nearly to the time consumed in the case of using single layer CNN simulator. Since all black-block or all white-block will be simulated one time, after convergence all final states were saved in the memory. Every time all black-block or all white-block encountered again , their final states are just restored from the memory since there is no need for simulation again. The structure of time multiplexing CNN simulator is demonstrated in Fig. (3).

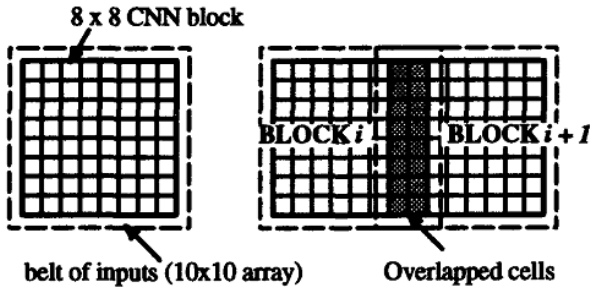


Figure 3: The structure of time multiplexing CNN simulator [9]

The steps of the time multiplexing CNN algorithm [9] with RK7(5) are summarized as follows:

1. Read input image
2. Divide the input image into a number of m blocks
3. Construct for each block, a belt of cells around it with size equal to the radius of the neighborhood from the input image
4. Between each two blocks construct an overlapped cells with size equal to the twice of the radius of the neighborhood
5. For each block, apply single layer CNN simulator [6]:
   (a) Read block C
   (b) Process all black-blocks or all white-blocks once and after convergence, store their state values in the memory
   (c) for each black or white block encountered again, just restore its state values from the memory
   (d) If the block didn't converge yet, compute its new state values using the equation (6) by applying RK7(5)
   (e) If the new state values are not the final states(didn't converge), go to (a)
   (f) Compute the error tolerance E_{ij}^A and E_{ij}^B for the templates A and B respectively for block C.
6. End

The most important point of the above steps will be the proposed numerical integration algorithm RK7(5) that will be presented in section (4.1). The main purpose of this algorithm is to reduce the CPU time or simulation time and to improve the edge detection results.

4 The numerical integration methods

Euler, RK-Gill [10] and RK-Butcher [1, 2] are three numerical integration methods used in the literature to solve the non linear differential equations which represent CNN dynamics. Section (4.1) discussed the proposed numerical integration algorithm RK7(5).

4.1 The numerical integration method RK7(5)

RK7(5) [12] was proposed as a new technique used in the time multiplexing CNN simulator[9] to improve the performance of the time multiplexing CNN simulator and edge detection results. The algorithm is based on minimizing the local truncation error and choosing an adaptive step size value using the selection step size algorithm. The equations of RK7(5) [12] are shown below:

$$k_1^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right)$$
$$k_2^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) + \frac{1}{18}k_1^{ij}$$
$$k_3^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) + \frac{1}{9}k_1^{ij}$$
$$k_4^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) + \frac{1}{24}k_1^{ij} + \frac{1}{8}k_2^{ij}$$
$$k_5^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) + \frac{2183971}{4000000}k_1^{ij} - \frac{8340813}{4000000}k_3^{ij} + \frac{3968421}{2000000}k_4^{ij}$$
$$k_6^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) - \frac{695768212}{746374411}k_1^{ij} - \frac{1803549175}{7007942496}k_3^{ij} + \frac{3474507053}{6790877290}k_4^{ij} + \frac{2188198899}{15264927763}k_5^{ij}$$
$$k_7^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) - \frac{118949348557}{8390623634}k_1^{ij} + \frac{53094780276}{9800512003}k_3^{ij} - \frac{8415376229}{2277049503}k_4^{ij} - \frac{18647567697}{10138317907}k_5^{ij} + \frac{275514944893}{11905950217}k_6^{ij}$$
$$k_8^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) - \frac{30828057951}{7654644085}k_1^{ij} - \frac{4511704}{324729}k_3^{ij} + \frac{16217851618}{1651177175}k_4^{ij} + \frac{282768186839}{40694064384}k_5^{ij} - \frac{104400780537}{15869257619}k_6^{ij} + \frac{5409241639}{9600177208}k_7^{ij}$$
$$k_9^{ij} = \tau f'\left(x_{ij}\left(n\tau\right)\right) - \frac{133775720546}{26753383835}k_1^{ij} + \frac{49608695511}{4066590848}k_3^{ij} - \frac{5989647201}{7901259813}k_4^{ij} - \frac{48035527651}{5727379426}k_5^{ij} + \frac{86266718551}{10188951048}k_6^{ij} - \frac{7751618114}{23575802495}k_7^{ij} + \frac{2289274942}{8464405725}k_8^{ij}$$

The solution of RK7(5) is given below:

$$x_{ij}\left(\left(n+1\right)\tau\right) = x_{ij}\left(n\tau\right) + \frac{597988726}{12374436915}k_1^{ij} + \frac{3138312158}{11968408119}k_4^{ij} + \frac{480882843}{7850665645}k_5^{ij} + \frac{988558885}{3512253271}k_6^{ij} + \frac{5302636961}{26425940286}k_7^{ij} + \frac{1259489433}{12163586030}k_8^{ij} + \frac{1016647712}{23899101975}k_9^{ij} \quad (9)$$

Our method is based on applying adaptive step size selection method [4] in time multiplexing CNN simulator using RK7(5).

The algorithm divides the input image into blocks and single layer CNN simulator is applied for each block. According to [4], The step size,h of $p(q)-$order RK methods is computed using the error per unit step (EPUS) [11] by using the following equation:

$$h_{n+1} = f_1 h_n \left(\frac{tolerance}{err_n}\right)^{\frac{1}{q+1}}, \quad (10)$$

Where f_1 is the factor of safety and $h_{n+1} = x_{n+1} - x_n$ is evaluated according to the estimation of err_n which is approximated using the equation[4]:

$$err_n \simeq \frac{y_n - \hat{y_n}}{h_n} \quad (11)$$

Since $y_n$and $\hat{y_n}$ are the estimated solutions of order pth and qth respectively.

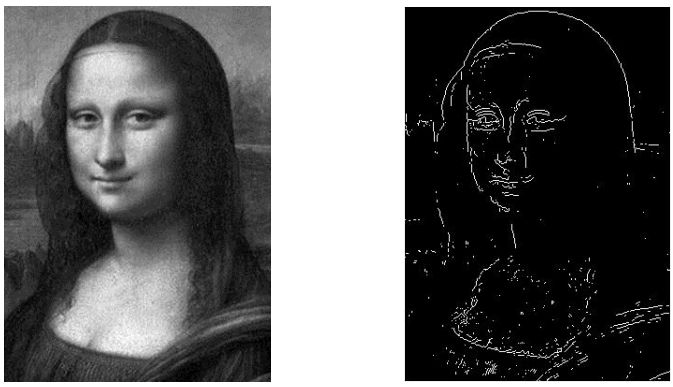
And $tolerance$ represents the required tolerance.

If $err_{n+1} \leq tolerance$, then y_{n+1} is applied otherwise, Eq. (10) is recalculated by $err_n \rightarrow err_{n+1}$.

So we propose Rk7(5) to get better results than those in the literature by reducing the computation time and improving the edge detection.

5 Simulation Results

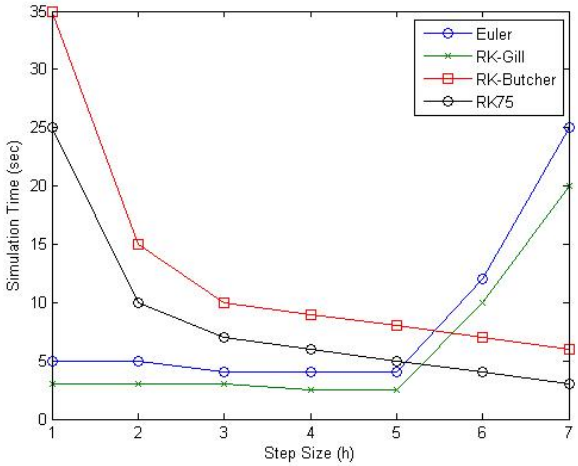
The input image which is used for testing is the Mona Lisa image. The edge detection results are shown in Fig. (4). The input image and the edge detection results are shown in Fig. (4a) and Fig. (4b) respectively.



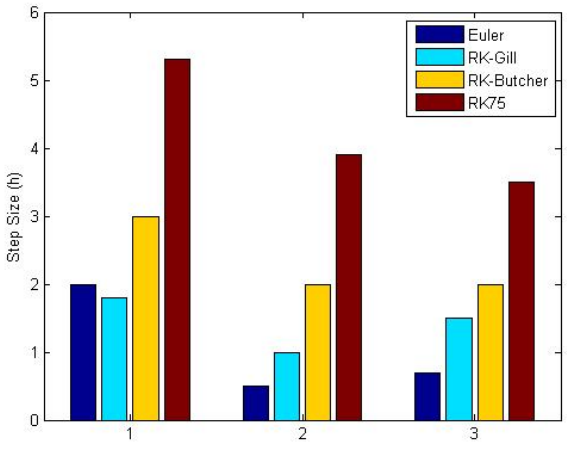
(a) Input image (b) Output image

Figure 4: Edge detection results

The performane results are shown in Fig. (5) since the simulation results are shown in Fig. (5a) and the maximum step size for all techniques are shown in Fig. (5b). The results obtained by simulating a small image of Mona Lisa for edge detection.



(a) Simulation time for 4 different techniques



(b) Maximum h for 4 different techniques:
1-Edge detection 2:Averaging template 3:Connected component

Figure 5: Performance results of the proposed method

From these results we notice that the value of the step size h using our method is higher than those in the related work. Hence the simulation time for the proposed method is decreased. . We notice that the value of h is considered to have a value to be increased to some range in order to avoid the divergence status. On the other hand, the value of h should be not very small because this implies more computation time as in the case of using Euler method.

Method	E^A	E^B	Local Truncation Error
Euler	0.785	0.885	0.7365
RK-Gill	0.676	0.764	0.6542
RK-Butcher	0.543	0.551	0.5754
RK 6(4)	0.329	0.428	0.457

Table 1: Results of the error tolerances E^A, E^B and Local Truncation Error

Table (1) demonstrates that the results of the error tolerances E^A and E^B and the local minimum truncation error using RK7(5) are less than those in the literature. The error tolerance represent the average of the error tolerances of the blocks of the input image. Also the simulation time is better than using RK-Gill and RK-Butcher methods found in the literature. So these results prove that our proposed method is promising when compared with those in the literature.

So the proposed method improved the results found in the literature by minimizing the error tolerances E^A and E^B and the local truncation error for each step which means that the obtained results using our proposed method has a good impact in decreasing the CPU time which is taken by the proposed simulator. In addition, the edge detection results were better if they were compared by the results found in the literature.

6 Conclusion

Our method used the selection step size algorithm to get better results using RK7(5). The time multiplexing CNN simulator using RK7(5) has shown better results for both the simulation time and edge detection results. This means that the performance of our method is better than those in the literature. Also, the obtained results of the error tolerances for the proposed method of time multiplexing CNN simulator is less than those found in the literature. For future work, we are planning to present more efficient numerical integration algorithms which can improve the performance of the simulator in addition to the quality of the output image for edge detection.

References

[1] M. Bader. A comparative study of new truncation error estimates and intrinsic accuracies of some higher order runge-kutta algorithms. *Computers and Biomedical Research Chemistry*, pages 121–124, 1987.

[2] M. Bader. A new technique for the early detection of stiffness in coupled differential equations and application to standard runge-kutta algorithms. *Theoretical Chemistry Accounts*, pages 215–219, 1988.

[3] L. O. Chua and L. Yang. Cellular neural networks: Theory and applications. *IEEE Trans. Circuits and Systems*, 1988.

[4] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick. Comparing numerical methods for ordinary differential equations. *Comparing numerical methods for ordinary differential equations*, 9:603–637, 1972.

[5] C.C. Lee and JP. de. Gyvez. Single-layer cnn simulator. In *IEEE International Symposium on Circuits and Sustems*. 1994.

[6] C.C. Lee and JP. de. Gyvez. Time-multiplexing cnn simulator. In *Proc. IEEE Int. Symposium on Circuits and Syst.*, pages 407–410, Dec. 1994.

[7] K. Murugesan, S. Sekar, V. Murugesh, and J.Y. Park. Numerical solution of an industrial robot arm control problem using the rk-butcher algorithm. *International Journal of Computer Applications in Technology*, pages 132–138, 2004.

[8] V. Murugesh and K. Murugesan. *Comparison of Numerical Integration in Raster CNN Simulation*, volume 3285. LNCS, Springer, 2004. 115-122.

[9] V. Murugesh and K. Murugesan. Simulation of time-multiplexing cellular neural networks with numerical integration algorithms. In *Lecture Notes in Computer Science*, volume 3991, pages 457–464, 2006.

[10] S.C. Oliveira. Evaluation of effectiveness factor of immobilized enzymes using rungekutta-gill method: how to solve mathematical undetermination at particle center point? *Bio Process Engineering*, 20, 1999.

[11] L. F. Shampine. Some practical runge-kutta formulas. *Math. Comp.*, 46:135–150, 1986.

[12] CH. Tsitouras and S. N. Papakostas. Cheap error estimation for runge–kutta methods. *SIAM J. Sci. Comput.*, 20(6):2067–2088, 1999.