Expert Systems with Applications 39 (2012) 12407–12417

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An efficient method for segmentation of images based on fractional calculus and natural selection

Pedram Ghamisi [a,*], Micael S. Couceiro [b,c], Jón Atli Benediktsson [d], Nuno M.F. Ferreira [c,e]

[a] Geodesy & Geomatics Engineering Faculty, K. N. Toosi University of Technology, Tehran, Iran
[b] Institute of Systems and Robotics, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal
[c] RoboCorp at the Electrical Engineering Department, Engineering Institute of Coimbra, Rua Pedro Nunes – Quinta da Nora, 3030-199 Coimbra, Portugal
[d] Faculty of Electrical and Computer Engineering, University of Iceland, Saemundargotu 2, 101 Reykjavik, Iceland
[e] GECAD – Knowledge Engineering and Decision Support Research Center Institute of Engineering, Polytechnic of Porto (ISEP/IPP) Porto, Portugal

ARTICLE INFO

Keywords:
Multilevel segmentation
Swarm Optimization
Image processing

ABSTRACT

Image segmentation has been widely used in document image analysis for extraction of printed characters, map processing in order to find lines, legends, and characters, topological features extraction for extraction of geographical information, and quality inspection of materials where defective parts must be delineated among many other applications. In image analysis, the efficient segmentation of images into meaningful objects is important for classification and object recognition. This paper presents two novel methods for segmentation of images based on the *Fractional-Order Darwinian Particle Swarm Optimization* (*FODPSO*) and *Darwinian Particle Swarm Optimization* (*DPSO*) for determining the n-1 optimal n-level threshold on a given image. The efficiency of the proposed methods is compared with other well-known thresholding segmentation methods. Experimental results show that the proposed methods perform better than other methods when considering a number of different measures.

1. Introduction

Image segmentation is the process of partitioning a digital image into multiple regions. In other words, image segmentation could assign a label to each pixel in the image such that pixels with the same label share certain visual characteristics. These objects contain more information than individual pixels since the interpretation of images based on objects is more meaningful than that based on individual pixels. Image segmentation is considered as an important basic task in the analysis and understanding of images, thus being widely used for further image processing purposes such as classification and object recognition (Sezgin & Sankur, 2004).

Image segmentation can be classified into four different types including texture analysis based methods, histogram thresholding based methods, clustering based methods and region based split and merging methods (Brink, 1995). One of the most common methods for the segmentation of images is the thresholding method, which is commonly used for segmentation of an image into two or more clusters (Kulkarni & Venayagamoorthy, 2010).

Thresholding techniques can be divided into two different types: optimal thresholding methods (Kapur, Sahoo, & Wong, 1985; Kittler & Illingworth, 1986; Otsu, 1979; Pun, 1980; Pun, 1981) and property-based thresholding methods (Lim & Lee, 1990; Tsai, 1995 & Yin & Chen, 1993). The former group search for the optimal thresholds which make the thresholded classes on the histogram reach the desired characteristics. Usually, it is made by optimizing an objective function. The latter group detects the thresholds by measuring some selected property of the histogram. Property-based thresholding methods are fast, which make them suitable for the case of multilevel thresholding. However the number of thresholds is hard to determine and needs to be specified in advance.

Several algorithms have been proposed in literature that addressed the issue of optimal thresholding (Brink, 1995; Cheng, Chen, & Li, 1998; Huang & Wang, 1995; Hu, Hou, & Nowinski, 2006; Kapur et al., 1985; Li, Zhao, & Cheng, 1995; Otsu, 1979; Pun, 1980; Saha & Udupa, 2001; Tobias & Seara, 2002; Yin & Chen, 1993). While many of them address the issue of bi-level thresholding, others have considered the multi-level problem. The problem of bi-level thresholding is reduced to an optimization problem to probe for the threshold t that maximizes the σ_B^2 and minimizes σ_W^2 (Kulkarni & Venayagamoorthy, 2010). For two level thresholding, the problem is solved by finding T^* which satisfies max

* Corresponding author. Address: No. 11, Fotouhi st., Farabi st., Ardakani st., Guyabadi st., Zafar st., Shariati st. Tehran, Iran Postal code: 1916759333. Tel.: +98 912 207 82 63.

E-mail addresses: p.ghamisi@gmail.com (P. Ghamisi), micaelcouceiro@isr.uc.pt, micael@isec.pt (M.S. Couceiro), benedikt@hi.is (J.A. Benediktsson), nunomig@isec.pt (N.M.F. Ferreira).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.04.078

$(\sigma_B^2(T^*))$ where $0 \leqslant T^* < L$ and L is the maximum intensity value. This problem could be extended to n-level thresholding through satisfying max σ_B^2 $(T_1^*, T_2^*, \ldots, T_{n-1}^*)$ that $0 \leqslant T_1^* < T_2^* < \cdots < T_{n-1}^* < L$. One way for finding the optimal set of thresholds is the exhaustive search method. The exhaustive search method based on the Otsu criterion (Otsu, 1979) is simple, but it has a disadvantage that it is computationally expensive (Kulkarni & Venayagamoorthy, 2010). Exhaustive search for $n-1$ optimal thresholds involves evaluations of fitness of $n(L-n+1)^{n-1}$ combinations of thresholds (Kulkarni & Venayagamoorthy, 2010) so this method is not suitable from a computational cost point of view. The task of determining $n-1$ optimal thresholds for n-level image thresholding could be formulated as a multidimensional optimization problem.

Alternative to the Otsu method, several biologically inspired algorithms have been explored in image segmentation (Fogel, 2000; Kulkarni & Venayagamoorthy, 2010; Lai & Tseng, 2004; Yin, 1999). Bio-inspired algorithms have been used in situations where conventional optimization techniques cannot find a satisfactory solution, for example when the function to be optimized is discontinuous, non-differentiable, and/or presents too many nonlinearly related parameters (Floreano & Mattiussi, 2008). The Particle Swarm Optimization (PSO) is a machine-learning technique loosely inspired by birds flocking in search of food (Kennedy & Eberhart, 1995). It basically consists of a number of particles that collectively move in the search space (*e.g.*, pixels of the image) in search of the global optimum (*e.g.*, maximizing the between-class variance of the distribution of intensity levels in the given image). However, a general problem with the PSO and other optimization algorithms is that of becoming trapped in a local optimum, such that it may work in some problems but may fail on others (Couceiro, Ferreira, & Machado, 2011).

The Darwinian Particle Swarm Optimization (*DPSO*) was formulated by Tillett et al. in 2005 (Tillett, Rao, Sahin, Rao, & Brockport, 2005) in search of a better model of natural selection using the *PSO* algorithm. In this algorithm, multiple swarms of test solutions performing just like an ordinary *PSO* may exist at any time with rules governing the collection of swarms that are designed to simulate natural selection. More recently, an extension of the *DPSO* using fractional calculus to control the convergence rate of the algorithm was presented by Couceiro et al. in 2011 (Couceiro et al., 2011), being denoted as fractional-order *DPSO* (*FODPSO*). The novel algorithm was successfully compared with both the fractional-order *PSO* from Pires, Machado, Oliveira, Cunha, and Mendes (2010) and the traditional *DPSO*.

Significant progress has been made in the creative inspiration of bio-inspired computer algorithms applied to optimization, estimation, control and many others through the application of principles derived from the study of biology (Floreano & Mattiussi, 2008). Santana, Alves, Correia, and Barata (2010) presented a swarm-based model for trail detection in real-time. Experimental results on a large dataset revealed the ability of the model to produce a success rate of 91% using a 20 Hz camera with a resolution of 640×480 that was carried through a scenario at an approximate speed of 1 m s^{-1}. The authors in Kulkarni and Venayagamoorthy (2010) compared the *PSO* and Bacteria Foraging algorithm (*BF*) with the Otsu method to determine the optimal threshold level for the deployment of sensor nodes. It should be noted that all methods were run offline and the *PSO* presented a superior performance when compared to the Otsu and the *BF*. Omran (2004) presented the application of the *PSO* to the field of pattern recognition and image processing. He introduced a clustering algorithm based on *PSO*. Further, he developed a dynamic clustering algorithm that could find the "optimum" number of clusters in a dataset with minimum user interference. Sathya and Kayalvizhi (2010) proposed a multilevel thresholding method based on *PSO* and compared their method with *GA*-based thresholding method. Results showed that the *PSO*-based image segmentation executed faster and was more stable than *GA*.

This paper mainly focuses on using one of the best performing *PSO* main variants (*cf.*, (Couceiro, Luz, Figueiredo, Ferreira, & Dias, 2010; Couceiro et al., 2011), created by Tillett et al. (2005)), denoted as *DPSO*, and the recently fractional-order extension, denoted as *FODPSO* (Couceiro et al., 2011). This is the first work to verify and apply the *FODPSO* and *DPSO* to multilevel segmentation. Bearing this idea in mind, the problem formulation of image n-level thresholding is presented in the following sub-sections. Section 2 presents a brief review of particle swarm algorithms, focusing on the strengths and weaknesses of the traditional *PSO*, the *DPSO* and the *FODPSO*. In Section 3, several images used to compare the *PSO*-based image segmentation variants with other commonly used algorithm such as genetic algorithms (*GA*) and *BF*. Finally, Section 4 outlines the main conclusions.

1.1. Image thresholding

Multilevel segmentation techniques provide an efficient way to perform image analysis. However, the automatic selection of a robust optimum n-level threshold has remained a challenge in image segmentation. This section presents a more precise formulation of the problem, introducing some basic notation.

Let there be L intensity levels in each *RGB* (red-green–blue) component of a given image and these levels are in the range $\{0,1,2,\ldots,L-1\}$. Then one can define:

$$p_i^C = \frac{h_i^C}{N}, \sum_{\substack{i=1 \\ C=\{R,G,B\}}}^{N} p_i^C = 1, \tag{1}$$

where i represents a specific intensity level, *i.e.*, $0 \leqslant i \leqslant L-1$, C represents the component of the image, *i.e.*, $C = \{R,G,B\}$, N represents the total number of pixels in the image and h_i^C denotes the number of pixels for the corresponding intensity level i in the component C. In other words, h_i^C represents an image histogram for each component C, which can be normalized and regarded as the probability distribution p_i^C. The total mean (*i.e.*, combined mean) of each component of the image can be easily calculated as:

$$\mu_T^C = \sum_{\substack{i=1 \\ C=\{R,G,B\}}}^{L} ip_i^C. \tag{2}$$

The 2-level thresholding can be extended to generic n-level thresholding in which $n-1$ threshold levels $t_j^C, j = 1,\ldots,n-1$, are necessary and where the operation is performed as expressed below:

$$F^C(x,y) = \begin{cases} 0, & f^C(x,y) \leqslant t_1^C \\ \frac{1}{2}\left(t_1^C + t_2^C\right), & t_1^C < f^C(x,y) \leqslant t_2^C \\ \vdots & \\ \frac{1}{2}\left(t_{n-2}^C + t_{n-1}^C\right), & t_{n-2}^C < f^C(x,y) \leqslant t_{n-1}^C \\ L, & f^C(x,y) > t_{n-1}^C \end{cases} \tag{3}$$

where x and y are the width (W) and height (H) pixel of the image of size $H \times W$ denoted by $f^C(x,y)$ with L intensity levels in each *RGB* component. In this situation, the pixels of a given image will be divided into n classes $D_1^C, \ldots, D_n^C$, which may represent multiple objects or even specific features on such objects (*e.g.*, topological features).

The simplest and computationally most efficient method of obtaining the optimal threshold is the one that maximizes the between-class variance which can be generally defined by:

$$\sigma_B^{C^2} = \sum_{\substack{j=1 \\ C=\{R,G,B\}}}^{n} w_j^C \left(\mu_j^C - \mu_T^C\right)^2, \tag{4}$$

where j represents a specific class in such a way that w_j^C and μ_j^C are the probability of occurrence and mean of class j, respectively. The probabilities of occurrence w_j^C of classes $D_1^C, \ldots, D_n^C$ are given by:

$$w_j^C = \begin{cases} \sum_{\substack{i=1 \\ C=\{R,G,B\}}}^{t_j^C} p_i^C, & j = 1 \\ \sum_{\substack{i=t_{j-1}^C+1 \\ C=\{R,G,B\}}}^{t_j^C} p_i^C, & 1 < j < n \\ \sum_{\substack{i=t_{j-1}^C+1 \\ C=\{R,G,B\}}}^{L} p_i^C, & j = n \end{cases}, \tag{5}$$

The mean of each class μ_j^C can then be calculated as:

$$\mu_j^C = \begin{cases} \sum_{\substack{i=1 \\ C=\{R,G,B\}}}^{t_j^C} \frac{i p_i^C}{w_j^C}, & j = 1 \\ \sum_{\substack{i=t_{j-1}^C+1 \\ C=\{R,G,B\}}}^{t_j^C} \frac{i p_i^C}{w_j^C}, & 1 < j < n \\ \sum_{\substack{i=t_{j-1}^C+1 \\ C=\{R,G,B\}}}^{L} \frac{i p_i^C}{w_j^C}, & j = n \end{cases}, \tag{6}$$

In other words, the problem of n-level thresholding is reduced to an optimization problem to search for the thresholds t_j^C that maximizes the three objective functions (*i.e.*, fitness function) of each *RGB* component, generally defined as:

$$\varphi^C = \max_{\substack{1 < t_1^C < \ldots < t_{n-1}^C < L \\ C=\{R,G,B\}}} \sigma_B^{C^2}\left(t_j^C\right). \tag{7}$$

Computing this optimization problem involves a much larger computational effort as the number of threshold levels increase. This brings us to the question: which kind of method should be used to solve this optimization problem for real-time applications?

Many methods have been proposed in the literature (Sezgin & Sankur, 2004). However, more recently, biologically inspired methods have been used as computationally efficient alternatives to analytical methods to solve optimization problems (Couceiro et al., 2010; Couceiro et al., 2010).

1.2. Efficiency evaluation

The computational time is one of the most important indicators along with fitness value which determine the ability of the algorithm. Provided that the data is large, the efficiency of the method is restricted to a great extent (Fan, Han, & Wang, 2009). For instance, remote sensing (*RS*) data, in particular hyperspectral images, are considerably large most of the time so using a high speed and efficient algorithm is highly preferable. Moreover, in real-time applications, using a high-speed algorithm is the main objective (Kulkarni & Venayagamoorthy, 2010). As a result, the evaluation of the *CPU* process time and fitness value seems vitally important to show the efficiency of the new method. In addition, since all bio-inspired methods are random and stochastic, the results are not completely the same in each run. Consequently, the stability of different methods should be evaluated by an appropriate index such as standard deviation value which will be described in Section 3.

PSO-based segmentation algorithms have been one of the most used in recent years. In fact, the traditional PSO-based segmentation has already been compared with GA-based algorithms or even exhaustive ones and has been found to present better results and being faster than both. In Hammouche, Diaf, and Siarry (2010), PSO-based segmentation method was superior compared to other algorithms such as *GA*, Differential Evaluation (*DE*), Ant Colony Optimization (*ACO*), Simulated Annealing (*SA*) and Tabu Search (*TS*) in terms of precision, robustness of the results and runtime. In Sathya and Kayalvizhi (2010), the authors show that *PSO* outperforms *GA* in terms of *CPU* time and fitness value for Kapur's and Otsu's functions. In Jiang, Luo, and Yang (2007), results show that *PSO*-family methods act better than *GA* with a learning operator (*GA-L*) in different measures. As a result, it is easy to detect that *PSO*-based segmentation methods are considered an efficient way in terms of finding optimal thresholds in less *CPU* process time. In Kulkarni and Venayagamoorthy (2010), *PSO* has been shown to be significantly faster than *BF* and exhaustive methods. As a result, comparison of the new methods with *PSO* in terms of *CPU* process time can be completely satisfying.

In this paper, two novel methods for segmentation of images based on DPSO and *FODPSO* are proposed. Both *DPSO* and *FODPSO* were introduced by Tillett et al. (2005) and Couceiro et al. (2011), respectively, for optimization of some primary test functions. We herein extend the concept of the algorithms to image segmentation. Therefore, they will be used to solve the Otsu problem for delineating multilevel threshold values. In other words, proposing a new thresholding based segmentation method which is robust in terms of the results and runtime, makes up our main goal. Further, in order to show the advantages of the new methods, we compare both algorithms with other algorithms that have been commonly used in the literature to determine the $n-1$ optimal n-level threshold on given images.

2. A brief review of the algorithms

The original *PSO* algorithm was developed by Eberhart and Kennedy in 1995 (Kennedy & Eberhart, 1995). The *PSO* basically takes advantage of the swarm intelligence concept, which is the property of a system whereby the collective behaviors of unsophisticated agents that are interacting locally with their environment, create coherent global functional patterns (Del Valle, Venayagamoorthy, Mohagheghi, Hernandez, & Harley, 2008). Imagine a flock of birds where each bird cries at an intensity proportional to the amount of food that it finds at its current location. At the same time each bird can perceive the position of neighboring birds and can tell which of the neighboring birds emits the loudest cry. There is a good chance that the flock will find a spot with the highest concentration of food if each bird follows a trajectory that combines three rules: (i) keep flying in the same direction; (ii) return to the location where it found the highest concentration of insects so far; and (iii) move toward the neighboring bird that cries the loudest (Kulkarni & Venayagamoorthy, 2010).

2.1. Particle Swarm Optimization (PSO)

In the traditional *PSO*, the candidate solutions are called particles. These particles travel through the search space to find an optimal solution, by interacting and sharing information with neighbor particles, namely their individual best solution (local best) and computing the neighborhood best. Also, in each step of the procedure, the global best solution obtained in the entire swarm is updated. Using all of this information, particles realize the loca-

Table 1
The *PSO* algorithm.

```
Initialize swarm (Initialize x_t^n, v_t^n, x̆_t^n, n̆_t^n and ğ_t^n)
Loop:
  for all particles
    Evaluate the fitness φ^C of each particle
    Update x̆_t^n, n̆_t^n and ğ_t^n
  Update v_t^n and x_t^n
  end
until stopping criteria (convergence)
```

tions of the search space where success was obtained, and are guided by these successes.

In each step of the algorithm (Table 1), a fitness function is used to evaluate the particle success. To model the swarm, each particle n moves in a multidimensional space according to position (x_t^n) and velocity (v_t^n) values which are highly dependent on local best $\left(\breve{x}_t^n\right)$, neighborhood best $\left(\breve{n}_t^n\right)$ and global best $\left(\breve{g}_t^n\right)$ information:

$$v_{t+1}^n = w v_t^n + \rho_1 r_1 \left(\breve{g}_t^n - x_t^n\right) + \rho_2 r_2 \left(\breve{x}_t^n - x_t^n\right) + \rho_3 r_3 \left(\breve{n}_t^n - x_t^n\right), \quad (8)$$

$$x_{t+1}^n = x_t^n + v_{t+1}^n. \quad (9)$$

The coefficients w, ρ_1, ρ_2 and ρ_3 assign weights to the inertial influence, the global best, the local best and the neighborhood best when determining the new velocity, respectively. Typically, the inertial influence is set to a value slightly less than 1. ρ_1, ρ_2 and ρ_3 are constant integer values, which represent "cognitive" and "social" components. However, different results can be obtained by assigning different influences for each component. For example, some methods do not consider the neighborhood best and ρ_3 is set to zero. Depending on the application and the characteristics of the problem, tuning these parameters properly will lead to better results. The parameters r_1, r_2 and r_3 are random vectors with each component generally a uniform random number between 0 and 1. The intent is to multiply a new random component per velocity dimension, rather than multiplying the same component with each particle's velocity dimension.

The particles in the *PSO* are evaluated for the fitness function, which is defined as the between-class variance σ_B^2 of the image-intensity distributions previously represented in (7).

In the beginning, the particles' velocities are set to zero and their position is randomly set within the boundaries of the search space. The search space will depend on the number of intensity levels L, *i.e.*, if the frames are 8-bit images then the particles will be deployed between 0 and 255.

The local, neighborhood and global bests are initialized with the worst possible values, taking into account the nature of the problem. The other parameters that need to be adjusted are population size and stopping criteria. The population size is very important to optimize to get an overall good solution in an acceptable time limit. Stopping criteria can be a predefined number of iterations without getting better results or other criteria, depending on the problem.

PSO reveals an effect of implicit communication between particles (similar to broadcasting) by updating neighborhood and global information, which affect the velocity and consequent position of particles. Also, there is a stochastic exploration effect due to the introduction of the random multipliers (r_1, r_2 and r_3).The *PSO* has been successfully used in many applications such as robotics (Couceiro, Luz, Figueiredo, & Ferreira, 2012; Pires, Oliveira, Machado, & Cunha, 2006; Tang, Zhu, & Sun, 2005), electric systems (Alrashidi & El-Hawary, 2006; Del Valle et al., 2008) and sports engineering (Couceiro et al., 2010).

2.2. Darwinian PSO

However, a general problem with the *PSO* and other optimization algorithms is that of becoming trapped in a local optimum such that it may work well on one problem but may fail on another problem. In order to overcome this problem many authors have suggested other adjustments to the parameters of the *PSO* algorithm combining fuzzy logic (*FAPSO*) where the inertia weight w is dynamically adjusted using fuzzy "*IF–THEN*" (Hammouche et al., 2010) rules or Gaussian approaches (*GPSO*) where the inertia constant w is no longer needed and the acceleration constants ρ_1, ρ_2 and ρ_3 are replaced by random numbers with Gaussian distributions (Jiang et al., 2007). More recently, Pires et al. used fractional calculus to control the convergence rate of the *PSO* (Sabatier et al., 2007). The authors rearrange the original velocity equation (8) in order to modify the order of the velocity derivative. Alternatively, many authors have considered incorporating selection, mutation and crossover, as well as the *DE*, into the *PSO* algorithm. The main goal is to increase the diversity of the population by either preventing the particles to move too close to each other and collide (Machado et al., 2010; Ortigueira & Tenreiro Machado, 2003) or to self-adapt parameters such as the constriction factor, acceleration constants (Podlubny, 1999), or inertia weight (Debnath, 2003).

The fusion between *GA* and the *PSO* originated the *GA-PSO* (Couceiro, Ferreira, & Machado, 2010) which combines the advantages of swarm intelligence and a natural selection mechanism, such as *GA*, in order to increase the number of highly evaluated agents, while decreasing the number of lowly evaluated agents at each iteration step. Similar to this last one, the *EPSO* is an evolutionary approach that incorporates a selection procedure to the original *PSO* algorithm, as well as self-adapting properties for its parameters. This algorithm adds a tournament selection method used in evolutionary programming (*EP*) (Pires et al., 2010). Based on the *EPSO*, a differential evolution operator has been proposed to improve the performance of the algorithm in two different ways. The first one (Yasuda, Iwasaki, Ueno, & Aiyoshi, 2008) applies the differential evolution operator to the particle's best position to eliminate the particles falling into local minima (*DEPSO*) while the second one (Venter & Sobieszczanski-Sobieski, 2002) applies it to find the optimal parameters (inertia and acceleration constants) for the canonical *PSO* (*C-PSO*).

In search of a better model of natural selection using the *PSO* algorithm, the Darwinian Particle Swarm Optimization (*DPSO*) was formulated (Tillett et al., 2005), in which many swarms of test solutions may exist at any time. Each swarm individually performs just like an ordinary *PSO* algorithm with rules governing the collection of swarms that are designed to simulate natural selection. Despite the similarities between the *PSO* and *GA*s, like randomly generated population, fitness function evaluation, population update, search for optimality with random techniques and not guaranteeing success; *PSO* does not use genetic operators like crossover and mutation, thus

Table 2
The *DPSO* algorithm.

Main program loop	Evolve swarm algorithm
For each swarm in the collection	For each particle in the swarm
Evolve the swarm (Evolve Swarm Algorithm: right)	Update Particles' Fitness
Allow the swarm to spawn	Update Particles' Best
Delete "failed" swarms	Move Particle
	If swarm gets better
	Reward swarm: spawn particle: extend swarm life
	If swarm has not improved
	Punish swarm: possibly delete particle: reduce swarm life

not being considered an evolutionary technique. On the other hand, the Darwinian Particle Swarm Optimization (*DPSO*) extends the *PSO* to determine if natural selection (Darwinian principle of survival of the fittest) can enhance the ability of the *PSO* algorithm to escape from local optima. The idea is to run many simultaneous parallel *PSO* algorithms, each one a different swarm, on the same test problem and a

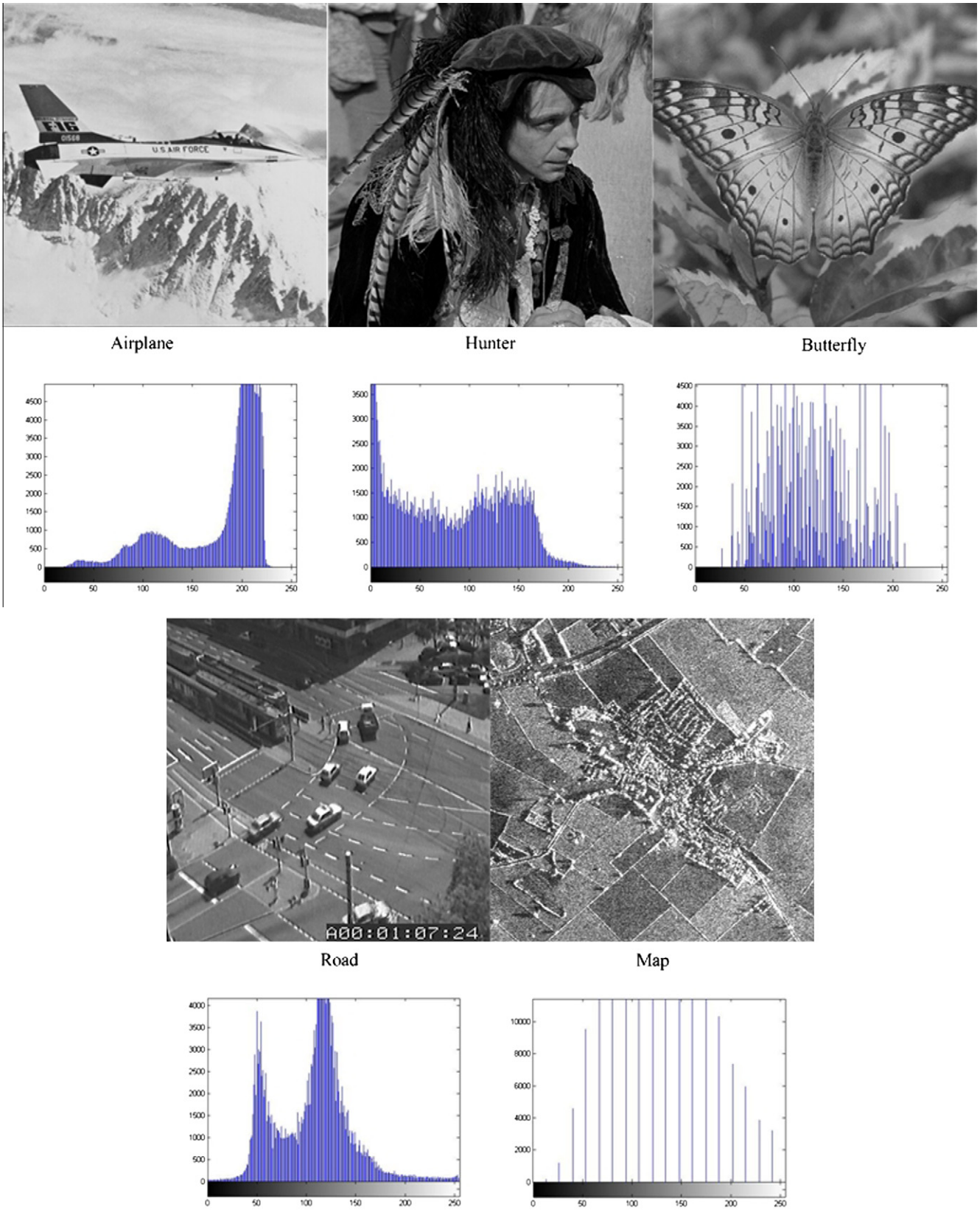

Fig. 1. Different test cases with their histograms.

Table 3
Initial parameters of the *PSO*, *DPSO* and *FODPSO*.

Parameter	PSO	DPSO	FODPSO
Num of Iterations	8	8	8
Population	200	30	30
ρ_1	1.5	1.5	1.5
ρ_2	1.5	1.5	1.5
W	1.2	1.2	1.2
$Vmax$	2	2	2
$Vmin$	−2	−2	−2
X_{max}	255	255	255
X_{min}	0	0	0
Min population	–	10	10
Max population	–	50	50
Num of swarms	–	4	4
Min swarms	–	2	2
Max swarms	–	6	6
Stagnancy	–	10	10
Fractional coefficient	–	–	0.75

simple selection mechanism is applied. When a search tends to a local optimum, the search in that area is simply discarded and another area is searched instead. In this approach, at each step, swarms that get better are rewarded (extend particle life or spawn a new descendent) and swarms which stagnate are punished (reduce swarm life or delete particles). To analyze the general state of each swarm, the fitness of all particles is evaluated and the neighborhood and individual best positions of each of the particles are updated. If a new global solution is found, a new particle is spawned. A particle is deleted if the swarm fails to find a fitter state in a defined number of steps (Table 2).

Some simple rules are followed to delete a swarm, delete particles, and spawn a new swarm and a new particle: (i) when the swarm population falls below a minimum bound, the swarm is deleted; and (ii) the worst performing particle in the swarm is deleted when a maximum threshold number of steps (search counter SC_C^{max}) without improving the fitness function is reached. After the deletion of the particle, instead of being set to zero, the counter is reset to a value approaching the threshold number, according to:

$$SC_C(N_{kill}) = SC_C^{max}\left[1 - \frac{1}{N_{kill}+1}\right], \tag{10}$$

where N_{kill} is the number of particles deleted from the swarm over a period in which there was no improvement in fitness. To spawn a new swarm, a swarm must not have any particle ever deleted and the maximum number of swarms must not be exceeded. Still, the new swarm is only created with a probability of $p = f/NS$, with f a random number in [0,1] and NS the number of swarms. This factor avoids the creation of newer swarms when there are large numbers of swarms in existence. The parent swarm is unaffected and half of the parent's particles are selected at random for the child swarm and half of the particles of a random member of the swarm collection are also selected. If the swarm initial population number is not obtained, the rest of the particles are randomly initialized and added to the new swarm. A particle is spawned whenever a swarm achieves a new global best and the maximum defined population of a swarm has not been reached. Like the *PSO*, a few parameters also need to be adjusted to run the algorithm efficiently: (i) initial swarm population; (ii) maximum and minimum swarm population; (iii) initial number of swarms; (iv) maximum and minimum number of swarms; and (v) stagnancy threshold. In estimation problems previously studied (Del Valle et al., 2008) and robotic exploration strategies developed (Alrashidi & El-Hawary, 2006), the *DPSO* has been successfully compared with the *PSO* showing a superior performance.

2.3. Fractional-Order Darwinian PSO

The *FODPSO* presented in Couceiro et al. (2011) is an extension of the *DPSO* in which fractional calculus is used to control the convergence rate of the algorithm. Fractional calculus (*FC*) has attracted the attention of several researchers (Machado et al., 2010; Ortigueira & Tenreiro Machado, 2003; Sabatier et al., 2007), being applied in various scientific fields such as engineering, computational mathematics, fluid mechanics, among others (Couceiro et al., 2010; Debnath, 2003; Pires et al., 2010; Podlubny, 1999). The discrete time implementation of the *Grünwald–Letnikov* definition based on the concept of fractional differential with $\alpha \in C$ of the signal $x(t)$, is given by:

$$D^{\alpha}[x(t)] = \frac{1}{T^{\alpha}}\sum_{k=0}^{r}\frac{(-1)^k \Gamma(\alpha+1)x(t-kT)}{\Gamma(k+1)\Gamma(\alpha-k+1)}, \tag{11}$$

Table 4
Average ± *STD* fitness values of different methods for different test cases.

Test image	Thresholds	FODPSO	DPSO	BF	PSO	GA
Airplane	2	1837.7974 ± 0.0140	1837.7787 ± 0.0298	1837.7517 ± 0.1021	1837.7222 ± 0.0796	1837.7144 ± 0.8355
	3	1911.6564 ± 0.0557	1911.5429 ± 0.1287	1910.7434 ± 1.5123	1905.7664 ± 1.2742	1844.5642 ± 2.0290
	4	1954.7374 ± 0.1624	1954.5612 ± 0.3566	1954.2480 ± 1.8991	1953.8872 ± 2.6057	1950.5919 ± 5.2334
	5	1979.6190 ± 0.2067	1978.7698 ± 0.5637	1978.4335 ± 2.1062	1977.9742 ± 3.1647	1973.0894 ± 7.4164
Hunter	2	3064.2066 ± 0.0292	3064.1684 ± 0.0470	3064.1188 ± 0.0322	3064.0688 ± 0.2534	3064.0156 ± 0.3781
	3	3213.2101 ± 0.1217	3212.9363 ± 0.1930	3213.4460 ± 0.9627	3212.0585 ± 1.5406	3211.7947 ± 2.0141
	4	3269.0574 ± 0.3526	3268.4573 ± 0.6478	3266.3504 ± 2.2936	3257.1767 ± 3.2342	3231.1313 ± 5.0298
	5	3307.5841 ± 0.7660	3305.6159 ± 1.6202	3291.1339 ± 3.6102	3276.3173 ± 4.1811	3244.7387 ± 9.7412
Butterfly	2	1553.0732 ± 0.0010	1553.0615 ± 0.0426	1553.0734 ± 0.0643	1553.0687 ± 0.0846	1552.4129 ± 1.5470
	3	1669.1929 ± 0.0706	1669.0419 ± 0.3586	1667.2801 ± 1.2113	1665.7589 ± 2.6268	1662.6963 ± 3.4022
	4	1710.6595 ± 0.4651	1709.9903 ± 0.6253	1707.0994 ± 2.2120	1702.9069 ± 3.7679	1696.6940 ± 5.3135
	5	1735.8941 ± 0.5968	1734.4957 ± 1.4495	1733.0317 ± 3.5217	1730.7879 ± 6.0747	1716.0428 ± 7.5842
Road	2	1321.3721 ± 0.0083	1321.3351 ± 0.0202	1321.3366 ± 0.0386	1321.1132 ± 0.0969	1320.7661 ± 0.8599
	3	1433.8901 ± 0.1036	1433.7674 ± 0.1643	1430.8712 ± 0.7123	1425.3853 ± 1.0941	1418.4695 ± 3.6425
	4	1490.1314 ± 0.2609	1489.5771 ± 0.9921	1488.2286 ± 1.9801	1483.1709 ± 2.8351	1476.7349 ± 5.8790
	5	1519.7184 ± 0.3494	1518.8896 ± 1.1087	1518.3493 ± 2.2354	1511.7474 ± 4.0074	1500.8104 ± 8.2580
Map	2	2340.3950 ± 0.0000	2340.3950 ± 0.0000	2340.3950 ± 0.0026	2340.3950 ± 0.0000	2252.3864 ± 1.2171
	3	2529.9384 ± 0.0000	2529.9384 ± 0.0000	2529.9348 ± 0.0548	2526.3034 ± 1.1180	2503.7932 ± 2.1368
	4	2621.1476 ± 0.0000	2621.1476 ± 0.0000	2621.1476 ± 0.3710	2618.4894 ± 2.9722	2617.9534 ± 3.7246
	5	2670.0640 ± 0.0000	2670.0640 ± 0.1969	2668.0699 ± 1.2189	2665.4116 ± 3.8519	2660.8599 ± 5.4901

Table 5
Average thresholds of different segmentation algorithm.

Image	Thresholds	FODPSO	DPSO	BF	PSO	GA
Airplane	2	116, 174	116, 174	117, 175	117, 174	116, 175
	3	93, 145, 190	95, 148, 193	91, 147, 190	99, 158, 193	86, 133, 204
	4	88, 132, 175, 203	88, 132, 174, 204	84, 127, 169, 202	84, 125, 168, 201	71, 119, 164, 200
	5	70, 106, 144, 180, 205	74, 109, 148, 181, 205	71, 110, 138, 175, 203	60, 101, 138, 177, 204	84, 124, 164, 188, 204
Hunter	2	51, 116	51, 116	51, 117	52, 116	51, 115
	3	35, 86, 134	35, 87, 134	36, 86, 135	39, 86, 135	36, 89, 133
	4	31, 72, 110, 146	26, 63, 102, 141	31, 80, 120, 152	36, 84, 130, 157	39, 93, 142, 163
	5	21, 53, 89, 122, 152	22, 52, 90, 118, 149	31, 73, 109, 141, 178	37, 85, 125, 154, 177	39, 94, 130, 169, 204
Butterfly	2	99, 151	98,151	99, 151	99, 150	100, 151
	3	82, 119, 159	81, 119, 160	78, 117, 162	79, 119, 164	74, 115, 155
	4	69, 98, 126, 162	71, 100, 130, 163	75, 105, 135, 165	80, 113, 145, 177	82, 119, 154, 184
	5	72, 98, 125, 150, 180	74, 103, 126, 153, 179	76, 104, 129, 154, 180	75, 106, 129, 157, 180	77, 107, 134, 171, 185
Road	2	90, 154	90, 154	90, 155	91, 155	90, 151
	3	86, 129, 186	86, 130, 187	89, 133,180	85, 127, 169	77, 121, 184
	4	72, 105, 135, 190	72, 106, 139, 193	78, 111, 139, 189	78, 114, 147, 205	74, 97, 139, 205
	5	69, 98, 124, 151, 202	73, 102, 124, 155, 205	70, 102, 128, 159, 211	71, 103, 134, 173, 225	79, 109, 142, 179, 204
Map	2	110, 186	113, 177	109, 176	113, 177	81, 173
	3	95, 148, 201	95, 140, 197	98, 146, 189	81, 145, 197	83, 132, 181
	4	81, 122, 169, 218	88, 122, 173, 224	88, 134, 173, 222	92, 133, 162, 206	90, 110, 158, 204
	5	80, 112, 140, 179, 218	76, 120, 145, 186, 221	80, 109, 135, 165, 224	79, 116, 139, 162, 204	68, 106, 138, 170, 214

where Γ is the gamma function, T is the sampling period and r is the truncation order.

An important property revealed by the *Grünwald–etnikov* is that while an integer-order derivative just implies a finite series, the fractional-order derivative requires an infinite number of terms. Therefore, integer derivatives are 'local' operators while fractional derivatives have, implicitly, a 'memory' of all past events. However, the influence of past events decreases over time.

The characteristics revealed by fractional calculus make this mathematical tool well suited to describe phenomena such as irreversibility and chaos because of its inherent memory property. In this line of thought, the dynamic phenomena of particle's trajectory configure a case where fractional calculus tools fit adequately.

Considering the inertial influence of (8) as $w = 1$, assuming $T = 1$ and similarly to Pires et al. (2010) work, the following expression can be defined:

$$D^{\alpha}\left[v_{t+1}^{n}\right] = \rho_1 r_1\left(\breve{g}_t^n - x_t^n\right) + \rho_2 r_2\left(\breve{x}_t^n - x_t^n\right) + \rho_3 r_3\left(\breve{n}_t^n - x_t^n\right). \quad (12)$$

Preliminary experimental tests in Couceiro et al. (2011) presented similar results for $r \geqslant 4$. Furthermore, the computational requirements increase linearly with r, *i.e.*, the *FODPSO* present a $\mathcal{O}(r)$ memory complexity. Hence, using only the first $r = 4$ terms of differential derivative given by (11) and (8) can be rewritten as (13):

$$v_{t+1}^{n} = \alpha v_t^n + \frac{1}{2}\alpha v_{t-1}^n + \frac{1}{6}\alpha(1-\alpha)v_{t-2}^n + \frac{1}{24}\alpha(1-\alpha)(2-\alpha)v_{t-3}^n + \rho_1 r_1\left(\breve{g}_t^n - x_t^n\right) + \rho_2 r_2\left(\breve{x}_t^n - x_t^n\right) + \rho_3 r_3\left(\breve{n}_t^n - x_t^n\right). \quad (13)$$

The *DPSO* is then considered as being a particular case of the *FODPSO* when $\alpha = 1$ (without 'memory'). Hence, the value of α greatly affect the inertial particles. With a small α, particles will ignore their previous activities, thus ignoring the system dynamics and being susceptible to get stuck in local solutions (*i.e.*, exploitation behavior). On the other hand, with a large α, particles will present a more diversified behavior which allows exploring new solutions, thus improving the long-term performance (*i.e.*, exploration behavior). However, if the exploration level is too high, then the algorithm may take too much time to find the global solution. Based on the experimental results from Couceiro et al. (2011), it will be used a fractional coefficient of $\alpha = 0.6$, thus resulting in a balance between exploitation and exploration.

3. Experimental results

DPSO- and *FOPSO*-based image segmentation which proposed in this paper was programmed in *MATLAB* on a computer having *Intel Core 2 Duo T5800* processor (2.00 *GHz*) and 3*GB* of memory. The proposed methods are tested on a few common images including: *Airplane*, *Hunter*, *Butterfly*, *Road* and *Map*. Fig. 1 illustrates different test cases along with the histograms of the images. The efficiency of the proposed methods is evaluated by comparing their results with a few popular methods such as *GA*, *BF* and *PSO*.

The *PSO*, *DPSO* and *FODPSO* methods are parameterized algorithms. Therefore, one needs to be able to choose the parameter values that would result in faster convergence (Table 3). The cognitive, social and inertial weights were chosen taking into account several works focusing on the convergence analysis of the traditional *PSO* (*cf.*, Jiang et al. (2007) and Couceiro et al. (2011)).

Table 6
The average CPU process time of different segmentation methods.

Test image	No. of thresholds	FODPSO (*S*)	DPSO (*S*)	PSO (*S*)
Airplane	2	0.4221	0.4382	0.4127
	3	0.4623	0.4844	0.4936
	4	0.5438	0.5516	0.6057
	5	0.5890	0.6065	0.7020
Hunter	2	0.3871	0.3966	0.3927
	3	0.4544	0.4761	0.4941
	4	0.5479	0.5517	0.5973
	5	0.5913	0.6031	0.6956
Butterfly	2	0.3661	0.3805	0.3854
	3	0.4483	0.4689	0.4925
	4	0.5038	0.5235	0.6018
	5	0.5731	0.5811	0.7001
Road	2	0.3674	0.3875	0.3903
	3	0.4569	0.4726	0.4896
	4	0.5071	0.5241	0.5949
	5	0.5680	0.5696	0.6930
Map	2	0.3542	0.3463	0.3828
	3	0.4214	0.4366	0.4858
	4	0.4907	0.5047	0.5877
	5	0.5589	0.5621	0.6869

3.1. Fitness evaluation

Since all the optimization methods are stochastic and random population-based, each of them runs 20 times and the average and standard deviation fitness values are brought in Table 4. All fitness values are calculated for 2, 3, 4, 5 thresholds. It is noteworthy that, despite small differences, all algorithms seem to reach the vicinities of the optimal solution, *i.e.*, higher between-class variance. Nevertheless, those differences are more evident in most situations as the number of thresholds increase. The *FODPSO* leads with a slightly higher fitness value than the *DPSO*.

Note that they both use natural selection in order to avoid stagnation. However, the *FODPSO* has a fractional order mechanism that allows controlling the convergence rate of particles, thus presenting a more exploiting behavior when near the solution vicinities. In regards to the differences of *PSO* family and *GA*, the *PSO*

Fig. 2. The result of segmentation with 2, 3, 4, 5 thresholds, respectively (from left to right).

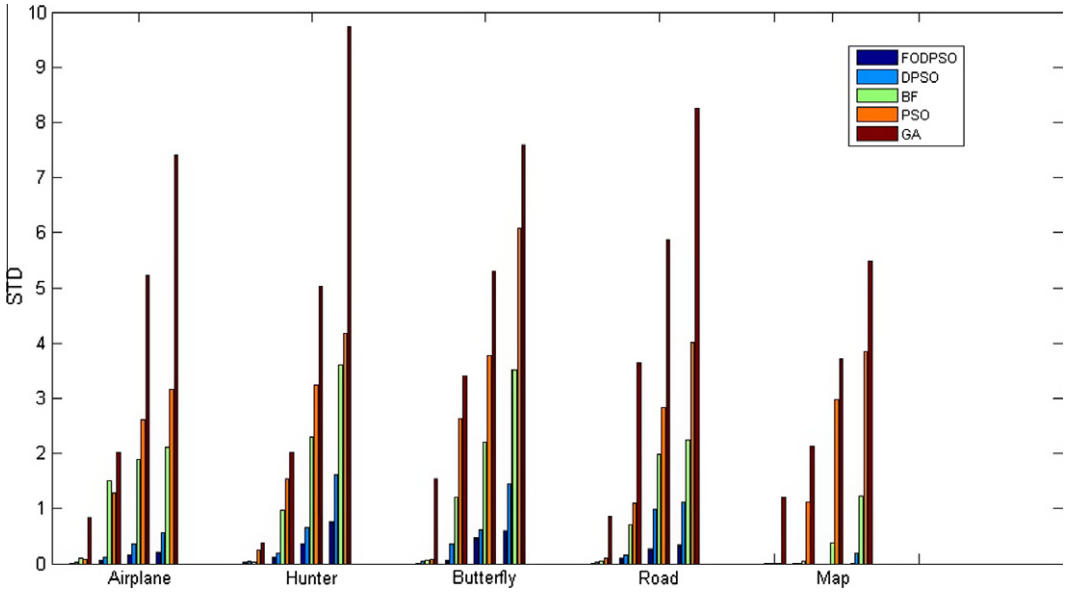

Fig. 3. Different *STD* values for different algorithms in face of different test cases (for each test case, 2, 3, 4, 5 thresholds, respectively (from left to right)).

family is an inherently continuous algorithm where as a *GA* is an inherently discrete algorithm (Venter & Sobieszczanski-Sobieski, 2002) and experiments conducted by Veeramachaneni, Peram, Mohan, and Osadciw (2003) showed that a *PSO* performed better than *GA*s when applied on some continuous optimization problems. In addition, *PSO* family was compared with a *GA* by Eberhart and Shi (1998) and Kennedy and Spears (1998). The results showed that *PSO* is generally faster and more robust to local solutions than *GA*s, especially when the dimension of a problem increases. As a result, when the number of dimension increases, a significant difference between the fitness values of the *PSO* family and *GA* happens and the *PSO* family shows better results than *GA* in higher dimensions. However, the tradition *PSO* suffers from premature convergence. Consequently, *BF* acts better than *PSO* in most cases. Table 5 demonstrates the optimal threshold values for the different methods.

3.2. CPU processing time

With regard to the *CPU* processing time, the *PSO* has been proven in the literature to require less *CPU* processing time for finding thresholds in comparison to *GA* and *BF* (*e.g.*, Sathya & Kayalvizhi, 2011). Therefore, we only compare the *CPU* time of *PSO*, *DPSO* and *FODPSO*. That brings us to Table 6 in which the *FODPSO* presents the best processing time, *i.e.*, it is able to reach its solution in less *CPU* time than *PSO* and *DPSO*. The *DPSO* still presents a lower *CPU* time than the *PSO* especially for higher threshold numbers. Nevertheless, it still needs more time to reach its solution than the *DPSO*. This is a small repercussion of having an exploitation activity when near the solution – a high level of exploitation allows a good short-term performance but slows down the convergence in order to reach a more feasible solution.

To visually compare the segmented results of different test cases by *FODPSO*, the segmented images with various threshold levels are given in Fig. 2. As can be seen from the figure, images with higher level of segmentation have more detail than the other images. In contrast, the 3 level segmented image is considered as the roughest image in different test cases. It is easy to conclude that by increasing the level of segmentation, the segmented image includes more detail. As a result, the 6-level segmented image in different test cases is smoother than the 3-level one.

3.3. Stability of different methods

Since almost all evolutionary methods are stochastic and random, the results are not completely the same in each run. Consequently, their results are affected by the nature and ability of the method. As a result, it seems necessary to evaluate the stability of the population-based algorithms. The comparison of the outputs gives us valuable information in terms of the stability of different algorithms, and which thresholding method is more suitable for segmentation of the images.

To evaluate the stability of the algorithm, the following index is used:

$$STD = \sqrt{\sum_{i=1}^{n} \frac{(\sigma_i - \mu)^2}{N}} \tag{14}$$

where *STD* is the standard deviation, σ_i is the best fitness value of the *i*th run of the algorithm, μ is the average value of σ and N is the repeated times of each algorithm ($N = 20$). It is easy to detect that the higher amount of *STD* represent more instability of the algorithm. The standard deviation of the different evolutionary algorithms for 20 runs, with 2, 3, 4, 5 thresholds are given in Table 4. From the Table 4, it can be seen that the *FODPSO* is the most stable evolutionary algorithms in comparison with others.

To improve the understanding of Table 4, Fig. 3 shows the standard deviation fitness values for the several algorithms in face of different test cases with different levels. As can be seen, in all experiments, *GA* shows the least stability among other bio-inspired method. According to the result, *FODPSO* is the most stable algorithm since illustrates the least *STD* values in comparison with other methods. *DPSO* and *BF* make up the second and third orders in terms of stability. In other words, the *FODPSO*-based segmentation is able to converge in approximately the same amount of time regardless on the image and the initial condition of particles.

Despite the observation that both *FODPSO* and *DPSO* present similar results, it is noteworthy that the fractional order algorithm is able to reach a slightly better fitness solution in less time. This should be highly appreciated as many applications require real-time segmentation methods such as the autonomous deployment of sensor nodes in a given environment or the detection of flaws in quality inspection of materials. In addition, *FODPSO* is slightly

faster than *DPSO* since fractional calculus is used to control the convergence rate of the algorithm. As described in Yasuda et al. (2008), a swarm behavior can be divided into two activities: (i) *exploitation*; and (ii) *exploration*. The first one is related with the convergence of the algorithm, thus allowing a good short-term performance. However, if the exploitation level is too high, then the algorithm may be stuck on local solutions. The second one is related with the diversification of the algorithm which allows exploring new solutions, thus improving the long-term performance. However, if the exploration level is too high, then the algorithm may take too much time to find the global solution. In the *DPSO*, the trade-off between exploitation and exploration can only be handled by adjusting the inertia weight. While a large inertia weight improves exploration activity, the exploitation may be improved using a small inertia weight. Since the *FODPSO* presents a fractional calculus strategy to control the convergence of particles with memory effect, the coefficient α allows providing a higher level of exploration while ensuring the global solution of the algorithm.

4. Conclusion

In this paper, two new methods for segmentation of images were proposed which is based on *Fractional-Order Darwinian Particle Swarm Optimization* (*FODPSO*) and *Darwinian Particle Swarm Optimization* (*DPSO*). The new methods were used for solving the Otsu problem for delineating multilevel threshold values and to overcome the disadvantages of previous evolutionary methods in terms of trapping in local optimum and high *CPU* process time. In this paper, the fitness value, *STD* and *CPU* process time were selected as the measures for comparing the output of different methods. Results indicate that *FODPSO* is more efficient than other methods in particular, when the level of segmentation increases, thus being able to find the better thresholds with more stability in less *CPU* processing time. As future research direction, the *FODPSO* will be evaluated in remote sensing applications and further compared with exhaustive methods. Moreover, due to the low computational complexity of the algorithm, a future direction may be the use of the *FODPSO* method in image segmentation applications for the real-time autonomous deployment and distributed localization of sensor nodes from an unmanned aerial vehicle (*UAV*).

References

Alrashidi, M. R., & El-Hawary, M. E. (2006). A survey of particle swarm optimization applications in power system operations. *Electric Power Component Systems, v34 i12*, 1349–1357.

Brink, A. D. (1995). Minimum spatial entropy threshold selection. *IEE Proceedings on Vision Image and Signal Processing, 142*, 128–132.

Cheng, H. D., Chen, J., & Li, J. (1998). Threshold selection based on fuzzy c-partition entropy approach. *Pattern Recognition, 31*, 857–870.

Couceiro, M. S., Luz, J. M. A., Figueiredo, C. M., Ferreira, N. M. F., & Dias, G. (2010). Parameter estimation for a mathematical model of the golf putting. In *WACI'10 – Proceedings of workshop applications of computational intelligence ISEC-IPC, December 2, Coimbra, Portugal* (pp. 1-8).

Couceiro, M. S., Ferreira, N. M. F., & Machado, J. A. T. (2011). In *Fractional order Darwinian Particle Swarm Optimization, FSS'11 – Symposium on fractional signals and systems, November 4–5, Coimbra, Portugal*.

Couceiro, M. S., Ferreira, N. M. F., & Machado, J. A. T. (2010). Application of fractional algoritms in the control of a robotic bird. *Journal of Communications in Nonlinear Science and Numerical Simulation – Special Issue, 15*(4), 895–910.

Couceiro, M. S., Luz, J. M. A., Figueiredo, C. M., & Ferreira, N. M. F. (2012). Modeling and control of biologically inspired flying robots. *Robotica* (Vol. 30, pp. 107–121). Cambridge University Press. 1.

Debnath, L. (2003). Recent applications of fractional calculus to science and engineering. *International Journal of Mathematics and Mathematical Sciences, 54*, 3413–3442.

Del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Hernandez, J. C., & Harley, R. G. (2008). Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation, 12*(2), 171–195.

Eberhart, R., & Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the seventh annual conference on evolutionary programming* (pp. 611–619). Springer-Verlag.

Fan, J., Han, M., & Wang, J. (2009). Single point iterative weighted fuzzy C-means clustering algorithm for remote sensing image segmentation. *Pattern Recognition, 42*, 2527–2540.

Floreano, D., & Mattiussi, C. (2008). *Bio-inspired artificial intelligence: Theories, methods, and technologies*. Cambridge, MA: MIT Press.

Fogel, D. B. (2000). *Evolutionary computation: Toward a new philosophy of machine intelligence* (Second ed.). Piscataway, NJ: IEEE Press.

Hammouche, K., Diaf, M., & Siarry, P. (2010). A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem. *Engineering Applications of Artificial Intelligence, 23*, 676–688.

Huang, L. K., & Wang, M. J. (1995). Image thresholding by minimizing the measure of fuzziness. *Pattern Recognition, 28*, 41–51.

Hu, Q., Hou, Z., & Nowinski, W. (2006). Supervised range-constrained thresholding. *IEEE Transactions on Image Processing, 15*, 228–240.

Jiang, M., Luo, Y. P., & Yang, S. Y. (2007). Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters, 102*(1), 8–16.

Kapur, J. N., Sahoo, P. K., & Wong, A. K. C. (1985). A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision Graphics Image Processing, 2*, 273–285.

Kennedy, J., & Eberhart, R. (1995). A new optimizer using particle swarm theory. In *Proceedings of the IEEE sixth international symposium on micro machine and human science* (pp. 39–43).

Kennedy, J., & Spears, W. (1998). Matching algorithms to problems: An experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *IEEE international conference on evolutionary computation, Achorage, Alaska, USA*.

Kittler, J., & Illingworth, J. (1986). Minimum error thresholding. *Pattern Recognition, 19*, 41–47.

Kulkarni, R. V., & Venayagamoorthy, G. K. (2010). Bio-inspired algorithms for autonomous deployment and localization of sensor. *IEEE Transactions on Systems, 40*(6), 663–675.

Kulkarni, R. V., & Venayagamoorthy, G. K. (2010). Bio-inspired algorithms for autonomous deployment and localization of sensor nodes. *IEEE Transactions, SMC-C40*(6), 663–675.

Lai, C. C., & Tseng, D. C. (2004). A hybrid approach using Gaussian smoothing and genetic algorithm for multilevel thresholding. *International Journal of Hybrid Intelligent Systems, 1*(3), 143–152.

Lim, Y. K., & Lee, S. U. (1990). On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques. *Pattern Recognition, 23*, 935–952.

Li, X., Zhao, Z., & Cheng, H. D. (1995). Fuzzy entropy threshold approach to breast cancer detection. *Information Sciences, 4*, 49–56.

Machado, J. A. T., Silva, M. F., Barbosa, R. S., Jesus, I. S., Reis, C. M., Marcos, M. G., et al. (2010). Some applications of fractional calculus in engineering. *Hindawi Publishing Corporation Mathematical Problems in Engineering*, 1–34.

Omran, M. G. H. (2004). Particle swarm optimization methods for pattern recognition and image processing. PhD Thesis, University of Pretoria, Pretoria.

Ortigueira, M. D., & Machado, J. A. T. (2003). Special Issue on fractional signal processing. *Signal Process, 83*, 2285–2480.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, Cybernetics, SMC-9*, 62–66.

Pires, E. J. S., Oliveira, P. B. M., Machado, J. A. T., & Cunha, J. B. (2006). Particle Swarm Optimization versus genetic algorithm in manipulator trajectory planning. In *7th Portuguese conference on automatic control, September 11–13*.

Pires, E. J. S., Machado, J. A. T., Oliveira, P. B. M., Cunha, J. B., & Mendes, L. (2010). Particle swarm optimization with fractional-order velocity. *Journal on Nonlinear Dynamics, 61*, 295–301.

Podlubny, I. (1999). *Fractional differential equations. Mathematics in Science and Engineering* (Vol. 198). San Diego, California: Academic Press.

Pun, T. (1980). A new method for grey-level picture thresholding using the entropy of the histogram. *Signal Processing, 2*, 223–237.

Pun, T. (1981). Entropy thresholding: A new approach. *Computer Vision Graphics Image Processing, 16*, 210–239.

Sabatier, J., Agrawal, O. P., & Tenreiro Machado, J. A. (Eds.). (2007). *Advances in Fractional Calculus - Theoretical Developments and Applications in Physics and Engineering*. Berlin: Springer. ISBN:978-1-4020-6041-0.

Saha, P. K., & Udupa, J. K. (2001). Optimum image thresholding via class uncertainty and region homogeneity. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 23*, 689–706.

Santana, P., Alves, N., Correia, L., & Barata, J. (2010). Swarm-based visual saliency for trail detection. In *Proceedings of the IEEE/RSJ 2010 international conference on intelligent robots and systems (IROS 2010), Taiwan*.

Sathya, P. D., & Kayalvizhi, R. (2010). PSO based tsallistresholding selection procedure for image segmentation. *International Journal of Computer Applications, 5*(4), 39–46.

Sathya, P. D., & Kayalvizhi, R. (2011). Modified bacterial foraging algorithm based multilevel thresholding for image segmentation. *Journal Engineering Applications of Artificial Intelligence, 24*(4).

Sezgin, M., & Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging, 13*(1), 146–168.

Tang, J., Zhu, J., & Sun, Z. (2005). A novel path panning approach based on appart and particle swarm optimization. In *Proceedings of the 2nd international symposium on neural networks, LNCS* (Vol. 3498, pp. 253–258).

Tillett, J., Rao, T. M., Sahin, F., Rao, R., & Brockport, S. (2005). Darwinian Particle Swarm Optimization. In *Proceedings of the 2nd Indian international conference on artificial intelligence* (pp. 1474–1487).

Tobias, O. J., & Seara, R. (2002). Image segmentation by histogram thresholding using fuzzy sets. *IEEE Transactions on Image Processing, 11*, 1457–1465.

Tsai, D. M. (1995). A fast thresholding selection procedure for multimodal and unimodal histograms. *Pattern Recognition Letters, 16*, 653–666.

Veeramachaneni, K., Peram, T., Mohan, C., & Osadciw, L. (2003). Optimization using particle swarm with near neighbor interactions. *Lecture notes computer science* (Vol. 2723). Springer-Verlag.

Venter, G., & Sobieszczanski-Sobieski, J. (2002). Particle swarm optimization. In *The 43rd AIAA/ASME/ASCE/AHA/ASC structures, structural dynamics and materials conference, Denver, Colorado, USA*.

Yasuda, K., Iwasaki, N., Ueno, G., & Aiyoshi, E. (2008). Particle swarm optimization: A numerical stability analysis and parameter adjustment based on swarm activity. *IEEE Transactions on Electrical and Electronic Engineering, 3*, 642–659.

Yin, P. Y. (1999). A fast scheme for optimal thresholding using genetic algorithms. *Signal Processing, 72*, 85–95.

Yin, P. Y., & Chen, L. H. (1993). New method for multilevel thresholding using the symmetry and duality of the histogram. *Journal of Electronics and Imaging, 2*, 337–344.