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

Pedram Ghamisi\textsuperscript{a,}\textsuperscript{*}, Micael S. Couceiro\textsuperscript{b,}\textsuperscript{c}, Jón Atlí Benediktsson\textsuperscript{d}, Nuno M.F. Ferreira\textsuperscript{c,e}

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

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, 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 $\sigma_1^2$ and minimizes $\sigma_2^2$ (Kulkarni & Venayagamoorthy, 2010). For two level thresholding, the problem is solved by finding $T^*$ which satisfies max
find the "optimum" number of clusters in a dataset with minimum multilevel thresholding method based on the application of the (2010) compared 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 \( n(L - n + 1)^{-1} \) 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 (Florenco & 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 (Florenco & 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 successful one. 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 \( \{0,1,2,...,L-1\} \). Then one can define:

\[
p_i^C = \frac{h_i^C}{N}, \quad \sum_{c=0}^{N} p_i^C = 1, \tag{1}
\]

where \( i \) represents a specific intensity level, i.e., \( 0 \leq i \leq L-1 \). \( C \) 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:

\[
p_i^C = \sum_{c=0}^{N} i p_i^C. \tag{2}
\]

The 2-level thresholding can be extended to generic \( n \)-level thresholding in which \( n - 1 \) threshold levels \( t_1^C, j = 1, ..., n - 1 \), are necessary and where the operation is performed as expressed below:

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

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 \( D_1^C, ..., D_n^C \), 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:
\[ \sigma^2_s = \sum_{j \in \{R, G, B\}} w^j \left( \mu^j - \mu^j_c \right)^2, \]  

\[ w^j = \begin{cases} \sum_{i=1}^{t^j} p^c_i, & j = 1 \\ \sum_{i=t^j+1}^{L} p^c_i, & 1 < j < n, \\ \sum_{i=t^j+1}^{L} p^c_i, & j = n \end{cases} \]  

where \( j \) represents a specific class in such a way that \( w^j \) and \( \mu^j_c \) are the probability of occurrence and mean of class \( j \), respectively. The probabilities of occurrence \( w^j \) of classes \( D_1^c, \ldots, D_n^c \) are given by:

The mean of each class \( \mu^j_c \) can then be calculated as:

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

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., 2011).

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, Dif, 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 Sathyaa and Kalyavizhi (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-
otions 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_n^t) and velocity (v_n^t) values which are highly dependent on local best (x^t), neighborhood best (\bar{n}_t) and global best (g_t^t) information:
\[ v_{n+1}^t = w v_n^t + \rho_1 (x^t - x_n^t) + \rho_2 (\bar{n}_n^t - x_n^t) + \rho_3 (g_t^t - x_n^t) \]
\[ x_{n+1}^t = x_n^t + v_{n+1}^t. \]

The coefficients w, \rho_1, \rho_2 and \rho_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, \rho_1, \rho_2 and \rho_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 \rho_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 \rho_1, \rho_2 and \rho_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 \sigma^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 (\rho_1, \rho_2 and \rho_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).

### Table 1
The PSO algorithm.

<table>
<thead>
<tr>
<th>Main program loop</th>
<th>Evolve swarm algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each swarm in the collection</td>
<td>For each particle in the swarm</td>
</tr>
<tr>
<td>Evolve the swarm (Evolve Swarm Algorithm: right)</td>
<td>Update Particles’ Fitness</td>
</tr>
<tr>
<td>Allow the swarm to spawn</td>
<td>Update Particles’ Best</td>
</tr>
<tr>
<td>Delete “failed” swarms</td>
<td>Move Particle</td>
</tr>
<tr>
<td></td>
<td>If swarm gets better</td>
</tr>
<tr>
<td></td>
<td>Reward swarm: spawn particle:</td>
</tr>
<tr>
<td></td>
<td>extend swarm life</td>
</tr>
<tr>
<td></td>
<td>If swarm has not improved</td>
</tr>
<tr>
<td></td>
<td>Punish swarm: possibly delete</td>
</tr>
<tr>
<td></td>
<td>particle: reduce swarm life</td>
</tr>
</tbody>
</table>

### 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” (Hammouch et al., 2010) rules or Gaussian approaches (GAPSO) where the inertia constant w is no longer needed and the acceleration constants \rho_1, \rho_2 and \rho_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 (Deb, 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 GAs, 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
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.
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 descendant) 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 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 \( x \in \mathbb{C} \) of the signal \( x(t) \), is given by:

\[
D^\alpha[x(t)] = \frac{1}{\Gamma(a)} \sum_{k=0}^{\infty} \left( -1 \right)^k \frac{t^k}{(\alpha + 1)^{k+1}} x(t-kT)
\]

where \( N_{\text{kill}} \) 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 \( p = f/N_S \), with \( f \) a random number in \([0,1]\) and \( N_S \) 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.

### Table 3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PSO</th>
<th>DPSO</th>
<th>FODPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Iterations</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Population</td>
<td>200</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>( \rho_1 )</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>( \rho_2 )</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>W</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Vmax</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Vmin</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( X_{\text{max}} )</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>( X_{\text{min}} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Min population</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Max population</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Num of swarms</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Min swarms</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Max swarms</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Stagnancy</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Fractional coefficient</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Test image</th>
<th>Thresholds</th>
<th>FODPSO</th>
<th>DPSO</th>
<th>BF</th>
<th>PSO</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airplane</td>
<td>2</td>
<td>1837.797 ± 0.0140</td>
<td>1837.787 ± 0.0298</td>
<td>1837.7517 ± 0.1021</td>
<td>1837.7222 ± 0.0796</td>
<td>1837.7144 ± 0.8355</td>
</tr>
<tr>
<td>Hunter</td>
<td>2</td>
<td>1911.6564 ± 0.0557</td>
<td>1911.5429 ± 0.1287</td>
<td>1910.7434 ± 1.5123</td>
<td>1905.7664 ± 1.7247</td>
<td>1844.5642 ± 2.0290</td>
</tr>
<tr>
<td>Butterfly</td>
<td>2</td>
<td>1553.0732 ± 0.0010</td>
<td>1553.0615 ± 0.0426</td>
<td>1553.0734 ± 0.0643</td>
<td>1553.0878 ± 0.0846</td>
<td>1553.1429 ± 1.5470</td>
</tr>
<tr>
<td>Road</td>
<td>2</td>
<td>1669.0920 ± 0.0010</td>
<td>1669.0419 ± 0.0358</td>
<td>1667.2801 ± 1.2113</td>
<td>1665.7589 ± 2.6268</td>
<td>1662.6963 ± 3.4022</td>
</tr>
<tr>
<td>Map</td>
<td>2</td>
<td>2430.3950 ± 0.0000</td>
<td>2430.3950 ± 0.0026</td>
<td>2430.3950 ± 0.0026</td>
<td>2430.3950 ± 0.0026</td>
<td>2425.3684 ± 1.2171</td>
</tr>
</tbody>
</table>

Average ± STD fitness values of different methods for different test cases.
where $\Gamma$ 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 $v^a_{t+1} = v^a_t + \frac{1}{r^+} x v^a_{t-1} + \frac{1}{6} x^2 v^a_{t-2} + \frac{1}{24} x^3 v^a_{t-3}$ (13):

$$D^a v^a_{t+1} = \rho_1 f_1 \left( g^a_t - x^a_t \right) + \rho_2 f_2 \left( x^a_t - x^a_* \right) + \rho_3 f_3 \left( n^a_t - x^a_* \right). \quad (12)$$

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

$$D^a v^a_{t+1} = \rho_1 f_1 \left( g^a_t - x^a_t \right) + \rho_2 f_2 \left( x^a_t - x^a_* \right) + \rho_3 f_3 \left( n^a_t - x^a_* \right). \quad (13)$$

The DPSO is then considered as being a particular case of the FODPSO when $x = 1$ (without 'memory'). Hence, the value of $x$ greatly affect the inertial particles. With a small $x$, 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 $x$, 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 $x = 0.6$, 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 3GB of memory. The proposed methods are tested on a few common images including: Airplane, Butterfly, Hunter, 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)).
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](image-url) The result of segmentation with 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 GAs 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 GAs, 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., Sathyia & 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:

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

where $STD$ is the standard deviation, $\sigma_i$ is the best fitness value of the $i$th run of the algorithm, $\mu$ is the average value of $\sigma$ 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


