Int J Thermophys (2014) 35:2374–2387
DOI 10.1007/s10765-014-1688-z

Cluster Segmentation of Thermal Image Sequences Using kd-Tree Structure

R. Świta · Z. Suszyński

Received: 27 October 2013 / Accepted: 4 July 2014 / Published online: 8 August 2014

Abstract This paper presents optimization methods for the K-means segmentation algorithm for a sequence of thermal images. Images of the sample response in the frequency domain to the thermal stimulation with a known spectrum were subjected to cluster segmentation, grouping pixels with similar frequency characteristics. Compared were all pixel characteristics in the function of the frame number and grouped using the minimal sum of deviations of the pixels from their segment mean for all the frames of the processed image sequence. A new initialization method for the K-means algorithm, using density information, was used. A K-means algorithm with a kd-tree structure C# implementation was tested for speed and accuracy. This algorithm divides the set of pixels to the subspaces in the hierarchy of a binary tree. This allows skipping the calculation of distances of pixels to some centroids and pruning a set of centroid clusters through the hierarchy tree. Results of the segmentation were compared with the K-means and FCM algorithm MATLAB implementations.

Keywords kd-tree · KKZ · K-means · Seeding · Thermal image sequences

1 Introduction

As a result of thermal excitation, a solid gives us a temperature response that we can study in both time and frequency domains. The sequence of thermal images includes in successive frames various information, depending on the internal structure of the sample. Analysis of thermal image sequences provides information about the quality of a sample's inner layers, its defects, and changes of thermal parameters. It can be

R. Świta (✉) · Z. Suszyński
Multimedia Systems and Artificial Intelligence Faculty in Technical University in Koszalin,
Śniadeckich street 2, 75-453 Koszalin, Poland
e-mail: robert.swita@wp.pl

used for detection of very small substance concentrations, study of photochemical reactions (e.g., photosynthesis), or obtaining depth profiles of samples in thermal microscopy.

Changes in the thermal sequence frames are continuous, and we can talk about the characteristics of each pixel in the sequence. To reduce processing, one can use cluster segmentation, which has a well-defined error and allows for an accurate quality comparison of various algorithms used for this purpose. This feature is quite rare among segmentation methods.

This paper presents a method of optimizing a K-means algorithm for segmentation of thermal image sequences using a kd-tree structure, which divides a set of the pixels to subspaces in the hierarchy of a binary tree. This allows skipping of the pixels distance calculation to some centroids and pruning clusters from the set of clusters during a search of the hierarchy tree. A new method of K-means algorithm initialization, using information about the pixels density in a sequence, was applied. Results of the speed and accuracy of the segmentation algorithm were compared with the implementation of the K-means algorithm and FCM algorithm implementations in MATLAB.

2 Principles of Algorithms

2.1 Description of Cluster Segmentation Algorithms

The fastest and most popular cluster segmentation algorithm is K-means (Forgy [1], MacQueen [2], Lloyd [3], Wu et al. [4]). Image clustering using K-means is a pixel segmentation algorithm, depending only on pixel values in consecutive frames and not taking into account locations and values of the neighbor pixels. The result of the segmentation consists generally of k incoherent objects, composed of pixels that are not necessarily located in their neighborhood, but of similar values. The procedure for the segmentation of an image sequence determines the sets of similar pixel value characteristics in a function of the frame number. From now on, for simplicity, we will in this article understand the term pixel as a vector of pixel values in successive frames of the sequence. The input parameter of the procedure is a set of k vectors of initial clusters. To eliminate the possibility of assigning two similar pixels to two different clusters, after assignment of pixels to the nearest segments, cluster centroids are recalculated to the mean value from all the pixels belonging to the cluster, and the assignment step is repeated (Lloyd algorithm). The iteration is interrupted when the pixels cease to change their assignment to clusters. The K-means algorithm is always convergent if the measure of the distance vectors is the Euclidean metric L_2, but the speed of convergence and accuracy of local solutions is determined mainly by the choice of the initial vector clusters (Pena et al. [5]). The standard solution is to choose centroids at random with a uniform probability distribution from all the values in the set (Forgy approach [1]). A KKZ method (Katsavounidis et al. [6]) is also a fast seeding algorithm in which pixels farthest from existing centroids are chosen for the new centroids. This prevents dividing a cluster into several smaller ones in case of choosing centroids, which are close to each other. Unfortunately, this method is very

sensitive to outliers in a sequence. Arthur and Vassilvitskii in [7] present a K-means++ algorithm, in which the initial centroids are selected from a set of input pixels, assuming a pixel probability proportional to the distance from the found centroids. The method thus combines the use of statistics and analysis of the mutual distance of clusters. The most common initialization algorithms for K-means, however, do not take into account the density distribution of pixels. Use of such information and the kd-tree structure is presented by Redmond and Heneghan [8]. Determination of the density distribution requires calculation of the mutual distances between pixels, which needs $O(n^2)$ count of operations. For a pixel k, its density can be determined, for example, by using the sum of the inverses of the pixel distances to all other pixels. The distance to near pixels has a much greater influence on this function value than to far ones (similar to the inverse distance weighting function). Factor ε prevents the function from escaping to infinity for cases, when $x_i = x_k$.

$$\rho_k = \sum_{i=1}^{P} \frac{1}{(x_i - x_k)^2 + \varepsilon} = \frac{1}{\varepsilon} \sum_{i=1}^{P} \left[\frac{(x_i - x_k)^2}{\varepsilon} + 1 \right]^{-1}. \tag{1}$$

Factor ε corresponds to a resolution of recorded distances; hence, it should satisfy $\varepsilon \ll (x_i - x_k)^2$ for all i, except for the case: $x_i = x_k$.

Using a Taylor series expansion of the function $(t+1)^{-1} = 1 - t + t^2 - t^3 + \cdots$ and expanding every term in the sum (Eq. 1) for $t = \frac{(x_i - x_k)^2}{\varepsilon}$, we obtain

$$\rho_k = \frac{1}{\varepsilon} \left(P - \frac{1}{\varepsilon} \sum_{i=1}^{P} (x_i - x_k)^2 + \frac{1}{\varepsilon^2} \sum_{i=1}^{P} (x_i - x_k)^4 - \cdots + (-1)^n \frac{1}{\varepsilon^n} \sum_{i=1}^{P} (x_i - x_k)^{2n} \right). \tag{2}$$

The series converges only when $\varepsilon > (x_i - x_k)^2$ for each pair of i, k.

Let us denote the vector of mean values of all pixels as $\overline{x}$ and the *variance* of pixel's values as

$$variance = \frac{1}{P} \sum_{i=1}^{P} (x_i - \overline{x})^2. \tag{3}$$

The density (Eq. 1) could be estimated, taking distances to be equal to the fixed mean value d_k:

$$\rho_k = \sum_{i=1}^{P} \frac{1}{(x_i - x_k)^2 + \varepsilon} \cong \frac{P}{d_k + \varepsilon}. \tag{4}$$

Expanding both sides of the equation into a Taylor series,

$$\frac{1}{\varepsilon}\left(P-\frac{1}{\varepsilon}\sum_{i=1}^{P}(x_i-x_k)^2+\frac{1}{\varepsilon^2}\sum_{i=1}^{P}(x_i-x_k)^4-\cdots\right)$$
$$\cong\frac{1}{\varepsilon}\left(P-P\frac{d_k}{\varepsilon}+P\left(\frac{d_k}{\varepsilon}\right)^2-\cdots\right) \tag{5}$$

and comparing second terms of both sides, d_k can be evaluated as

$$d_k=\frac{1}{P}\sum_{i=1}^{P}(x_i-x_k)^2=\frac{1}{P}\sum_{i=1}^{P}(x_i-\overline{x})^2+(x_k-\overline{x})^2=variance+(x_k-\overline{x})^2. \tag{6}$$

Since ε should be as small as possible, we want to choose for its value the convergence limit for the series $\varepsilon\approx d_k$. Additionally, ε has to be the same for all pixels, so we take the mean value of all d_k:

$$\varepsilon=\frac{1}{P}\sum_{k=1}^{P}d_k=\frac{1}{P}\sum_{i=1}^{P}\left[variance+(x_k-\overline{x})^2\right]=2^{*}variance. \tag{7}$$

And finally we can estimate the pixel's k density as

$$\frac{1}{P}\rho_k=\frac{1}{3^{*}variance+(x_k-\overline{x})^2}. \tag{8}$$

The local pixel density approximation using Eq. 8 requires O(n) operations.

For the segment centroids, the pixels with a greater local density, but also more distant from the already found centers should be selected. In our experiments, as the first center, the pixel with the highest density is selected. Due to the less accurate estimation of the local density, the other centers of segments were chosen from pixels with a maximum product of the square root of the density and its distance from the already found centers. We denote such an initialization method as high density (HD) for the density described by Eq. 1 and its approximation using Eq. 8.

The segmentation algorithm Fuzzy C-Means (FCM, Bezdek [9]) assumes that the pixels may be partially assigned to a segment as defined by the membership function μ (soft membership). Equation 9 presents a modification of cluster centroids to the center of gravity of the segments for each frame, which takes place after iteration:

$$\bar{x}_j=\frac{\sum_{i=1}^{P}\mu(x_i,\bar{x}_j)x_i}{\sum_{i=1}^{P}\mu(x_i,\bar{x}_j)}. \tag{9}$$

Equation 10 defines the membership function for each pair of the ith pixel and jth segment:

$$\mu(x_i, \overline{x}_j) = \frac{(x_i - \overline{x}_j)^{-2/(r-1)}}{\sum_{j=1}^{S} (x_i - \overline{x}_j)^{-2/(r-1)}}, \tag{10}$$

where r is the fuzziness of the member function, $r \geq 1$ and S is the segment count.

The K-means algorithm is thus a special case of the FCM method, for which the membership function is binary, taking one for the pixel and assigned segment pair and zero for the pixel with other segments (hard membership). The FCM algorithm can find, in general, a better solution, resulting, however, with increased computational effort and hence, a longer processing time.

The standard implementation of the K-means algorithm calculates the distances between pixels and all the cluster centers. As long as the cluster centers change, distance calculations for pixels have to be evaluated, and pixels are assigned to the nearest segments. However, organizing the collection of pixels in a kd-tree structure, one can construct a K-means algorithm which assigns groups of pixels to the segment simultaneously, and not one segment to one pixel at a time. There is often no need for calculating distances from each pixel to the centers of each segment. Groups of pixels can be assigned to a segment, or rejected as located too far away. The input data are two-dimensional (pixel values as a function of the frame number). The kd-tree could be created by comparing the data alternately in these dimensions. More effective, however, is to sort characteristics, for example, on their mean values (reducing the data to a single dimension). A more accurate similarity indicator is the square length of the characteristic vector. Since the square length of the vector is always non-negative, in order to distinguish the characteristics of a negative mean value, it is assumed that the similarity index is the square length of the vector with the sign from the mean value. Such a classification groups similar characteristics in the nodes of the kd-tree, fewer comparisons during processing are needed, and the computation time is shorter. Equation 11 shows how to determine the characteristic square length *Len* of the pixel vector v (with a dot product), by which pixels are sorted. Vector *ones* has coordinates all equal to 1.

$$Len = \text{sign}(v \circ ones) * v \circ v \tag{11}$$

On a sorted pixels basis, a binary kd-tree is created. The leaves of the tree correspond to the pixels.

The new nodes are created from two nodes of the lower level, starting from the lowest level of the leaves. In each node the maximum distance of its children from the center is stored—the node deviation. During segmentation, the kd-tree is searched recursively using a filtering procedure. The parameter for the procedure is a list of clusters, in which segments located too far away from the center of the node are pruned (Fig. 1). The triangle inequality for distances is used. The distance between a segment and any node's pixel is less than a sum of distance Δ from a segment to a node and the node's deviation d, but greater than the difference between them. The segment is close enough to the node's pixels if its minimum distance to the pixels will be smaller than the maximum distance between the pixels of the current node and its

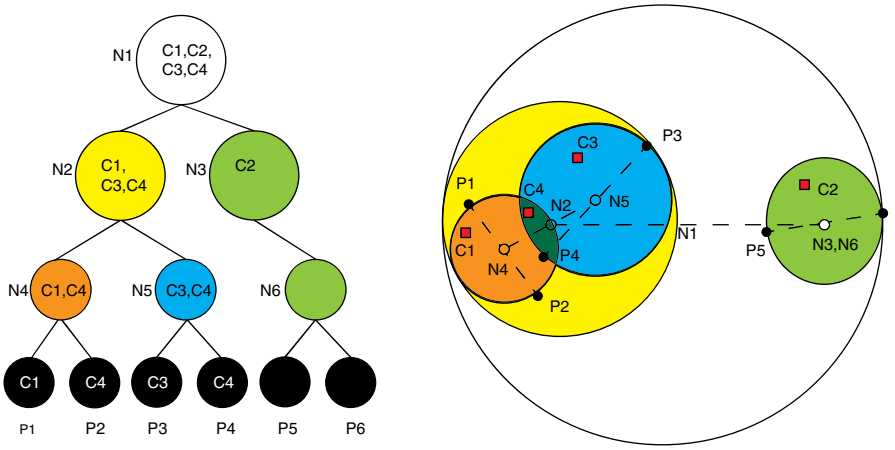

Fig. 1 Building pixel hierarchy in binary kd-tree

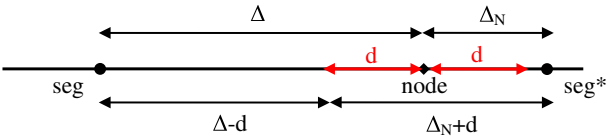

Fig. 2 Using triangle inequality for pruning segment centroids during kd-tree search

nearest segment. In Eq. 12 Δ_N is the distance from the node to its nearest segment (Fig. 2);

$$\begin{aligned} \Delta - d &< \Delta_N + d, \\ \Delta &< \Delta_N + 2d. \end{aligned} \tag{12}$$

Pseudocode of filtering procedure is presented below:

```
Filter(Node, Segments)
{
    seg* <- closest segment to Node
    foreach (seg in Segments)
        if ( Node.DistanceTo(seg) > Node.DistanceTo(seg*) + 2 * Node.Deviation)
            Segments = Segments \ {seg};
    if (Segments == {seg*} || Node.Children == {})
    {
        seg*.CentroidSum += Node.CentroidSum;
        seg*.CentroidCount += Node.CentroidCount;
    }
    else
        foreach (child in Node.Children)
            Filter(child, Segments);
}
```

The node-to-segment assignment involves updating the sum of segment centroids and segment centroids count by summing with the node's respective properties. This enables a later calculation of the segment centroid without aggregating error.

When using the Euclidean norm L_2, taking a triangle inequality into account requires the calculation of square roots of vector squared lengths (inequality for the squared lengths may not be satisfied). Such problems are not posed by city metric L_1 (Manhattan, taxi), but K-means using L_1 and the algorithm is not always convergent. The proper method would determine centers of segments based on the median pixels rather than the mean (K-median algorithm). We do not use correlation in our calculation either, because correlation indicates the degree of similarity between the characteristics, but does not allow determination of the characteristic most similar to all others characteristics of pixels in the segment.

In general, during segmentation, some segments can eventually lose all their pixels. If we choose for the initial centers pixel values from the sequence, it will mean that some of the selected pixels must have identical characteristics. The algorithms implemented in MATLAB use two strategies—ignore (drop) and selecting for the center of the empty segment a pixel which is farthest from its center (singleton). In our implementation of the K-means, if the initial centers of segments are selected from pixels in sequence, none of the segments ever lose all their pixels, because at least one pixel is at zero distance from a segment's center, and a pixel check of the segment-to-pixel distance starts from the currently assigned segment. In this way we eliminate the need for special processing of empty segments. This technique is also used in an algorithm with the kd-tree.

2.2 Used Algorithms and Techniques

Clustering algorithms in MATLAB were performing a batch update phase only, without trying to improve the result with a time-consuming on-line phase. In the batch update mode, if the segment has not lost pixels, or gained new ones, calculations for its distance to the pixels were not repeated. In this way, successive iterations are performed faster and the segmentation process is significantly shortened. In our implementation, in addition to memorizing segments, whose centers have not been moved, the distances of pixels from the segments, which are known to be changed in the next iteration, are calculated only until reaching the excess of the minimum currently known distance between the pixel and the segment. Additionally, an algorithm was developed in which adding or removing pixels from a segment is reflected immediately, not after the iteration (dynamic update).

In the experiments we used algorithms K-means and FCM in MATLAB, object-oriented algorithms developed in C# with the update of the segments at the end of an iteration or with a dynamic update and the K-means algorithm using the kd-tree structure.

2.3 Segmentation Error

For segmentation of an image sequence, the absolute error for S objects is defined as the sum of deviations of all P pixels from the nearest centers of segments for all K frames of the sequence:

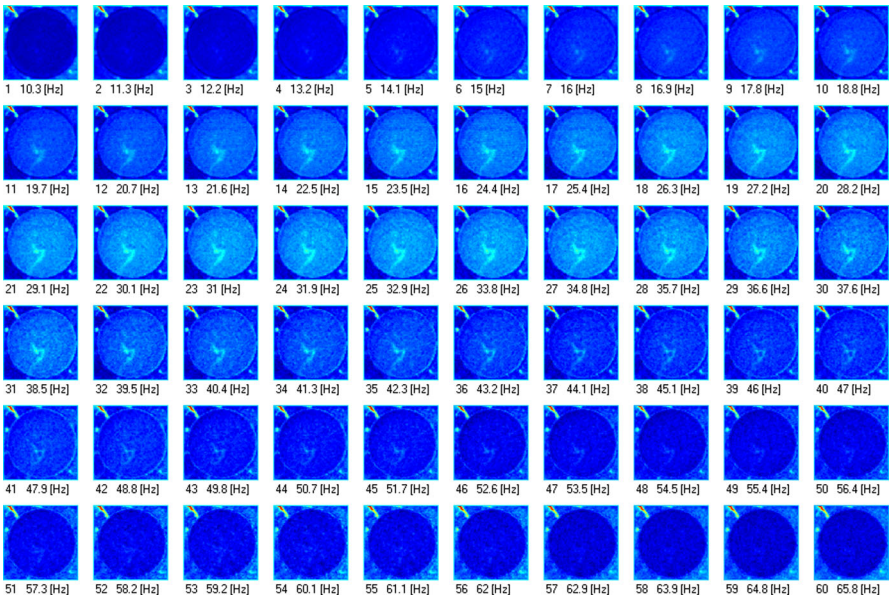

Fig. 3 Frames of Seq1 sequence

$$error = \sum_{k=1}^{K} \sum_{i=1}^{P} \min_{j \in <1,S>} \left(x_{ki} - \bar{x}_{kj}\right)^2. \tag{13}$$

The absolute error can also be presented as the sum of the absolute errors in individual segments and illustrated as the characteristic of the absolute segment errors in a function of the segment number:

$$error = \sum_{j=1}^{S} \sum_{k=1}^{K} \sum_{i=1}^{P_{\text{seg}(j)}} \left(x_{kji} - \bar{x}_{kj}\right)^2 = \sum_{j=1}^{S} error_{\text{seg}(j)}. \tag{14}$$

3 Results

We show the study of three sequences of thermal images with a resolution of 128×128 pixels and the number of frames from 1 to 512 (Fig. 3). The K-means algorithm has a very strong dependence on the choice of the initial centers of the segments. Therefore, in the first experiment, we compared the HD initialization method with Forgy's approach for 20 iterations, KKZ, and the HD approximation method. Comparison of methods for initializing the K-means algorithm is shown in Table 1. Selection of the initial centers determines the error of the first iteration. That error for this experiment was also given, as a small final error is related to finding a good initial estimate. A small initial error cannot guarantee, and is not sufficient to achieve a good end result, but as

Table 1 Comparison of initialization methods for K-means algorithm

Sequence	Seq1 (128 × 128 × 61)				Seq2 (128 × 128 × 512)				Seq3 (128 × 128 × 512)			
Init method	Start error (10^6)	Stop error (10^6)	Time (s)	Iter	Start error (10^6)	Stop error (10^3)	Time (s)	Iter	Start error (10^6)	Stop error (10^3)	Time (s)	Iter
FA (best from 20)	9.471	5.182	723	81	1.783	635.723	2235	66	1.747	637.888	3071	79
KKZ	15.668	5.264	26.5	88	1.964	614.920	162	147	2.233	631.565	148	127
HD approx	12.965	5.068	34.5	114	1.633	599.453	85,5	71	2.282	628.748	158	133
HD	9.552	4.935	270	54	0.919	508.404	1759	47	0.990	544.548	1773	41

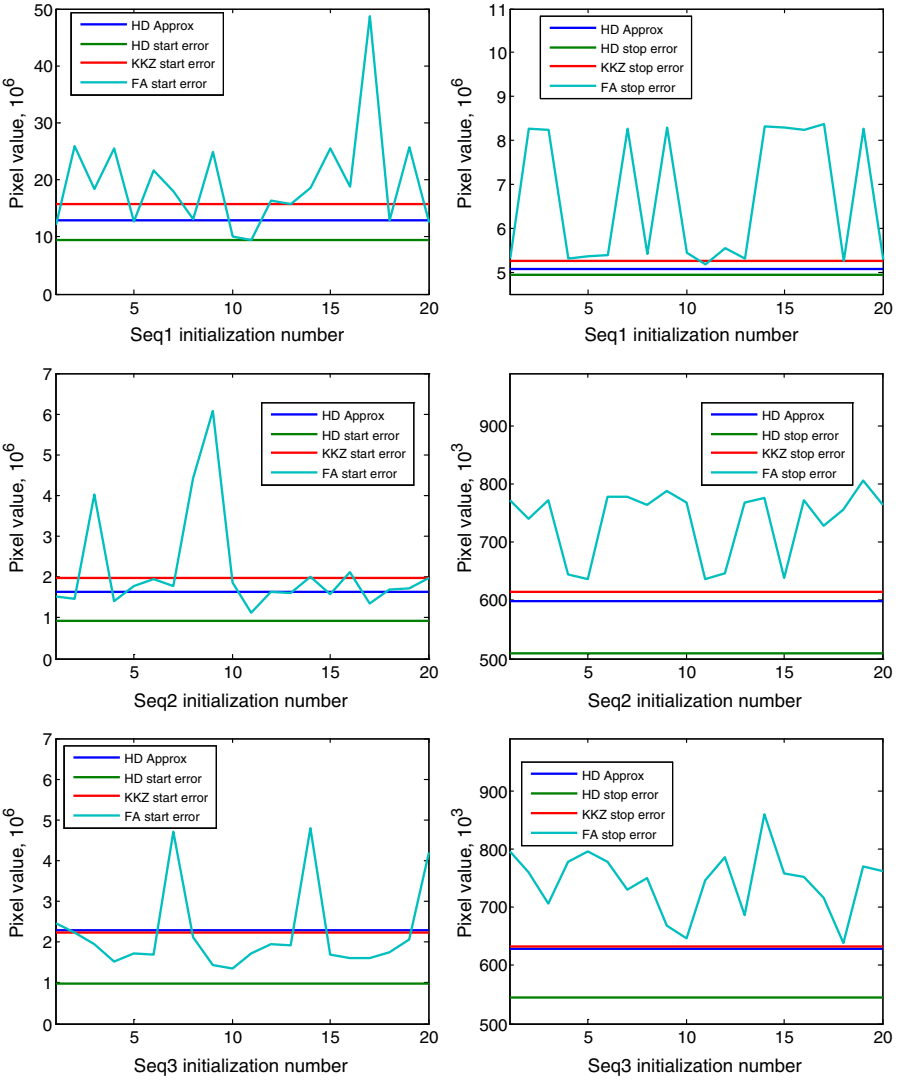

Fig. 4 Start and stop errors of K-means with different seeding algorithms for Seq1, Seq2, and Seq3

experiments reveal, it is a prerequisite. Results of different initialization methods are compared in Fig. 4 with the results of 20 consecutive random initializations. The HD initialization got the best results. An approximation for the HD method also obtained better results than for the KKZ and FA initializations. Due to the high computational complexity of the HD method, in further experiments, we used the HD approximation, calculating the density of pixels in accordance with Eq. 8. Prior to segmentation using the kd-tree, the pixels were sorted according to their characteristic length.

Table 2 present results of the segmentation of a Seq2 sequence for a different number of frames using all investigated K-means algorithms. Figure 5 illustrates

Table 2 Computing time and error measurements of cluster algorithms for different number of frames in Seq2

Algorithm	Seq2 (128 × 128 × 1)			Seq2 (128 × 128 × 2)			Seq2 (128 × 128 × 4)			Seq2 (128 × 128 × 8)			Seq2 (128 × 128 × 16)		
	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter
MATLAB KM	0.251	6.5	68	31.7	11	112	658	9.9	85	13448	7	46	30801	12.2	109
MATLAB FCM	0.532	170	200	30.3	207	200	682	224	200	12824	248	200	29933	337	200
C# KM	0.253	4	50	33.3	4	47	662	6.5	58	13918	6.5	67	32035	7.5	68
C# Dynamic KM	0.253	4	40	32.6	7	34	662	15	51	13806	15	55	31759	17	45
C# kd-Tree	0.251	1.3	62	30.8	37	149	653	60	125	13421	26.5	118	31280	48.5	109
Algorithm	Seq2 (128 × 128 × 32)			Seq2 (128 × 128 × 64)			Seq2 (128 × 128 × 128)			Seq2 (128 × 128 × 256)			Seq2 (128 × 128 × 512)		
	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter	Error	Time	Iter
MATLAB KM	57402	14.4	98	101606	24.4	158	181683	27	66	309551	91	184	573119	167	177
MATLAB FCM	55566	591	200	98186	889	200	175800	1560	200	336570	2762	200	673960	5582	200
C# KM	57872	7.5	40	98611	19	88	181406	31	91	334793	27	40	599453	85.5	71
C# Dynamic KM	57342	33.5	40	98730	51	67	179365	62	32	331827	91	23	601627	205	36
C# kd-Tree	57433	71	97	96487	101	85	175142	202	137	316563	551	181	573739	808	125

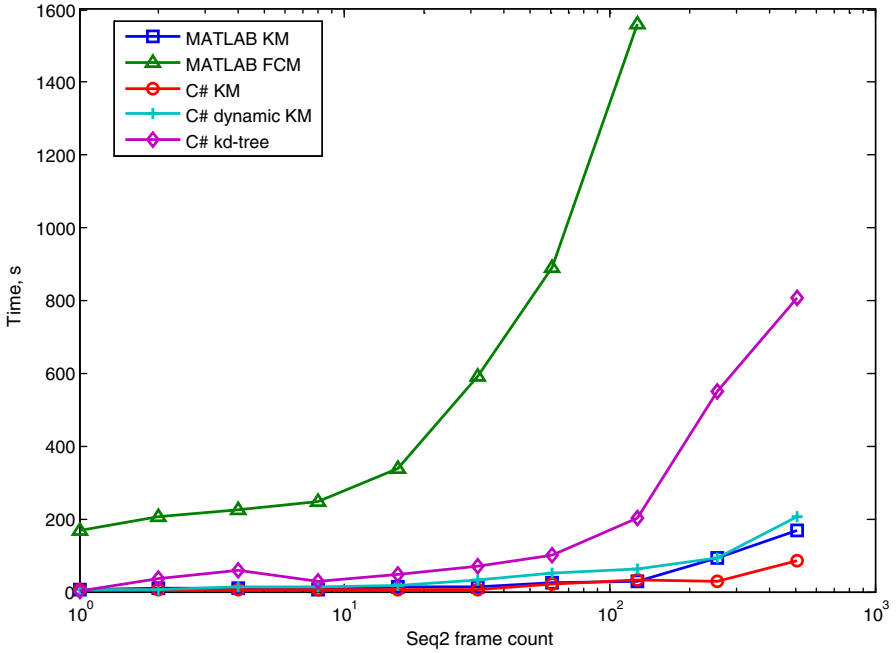

Fig. 5 Calculation time comparison for different K-means implementations in function of frames count

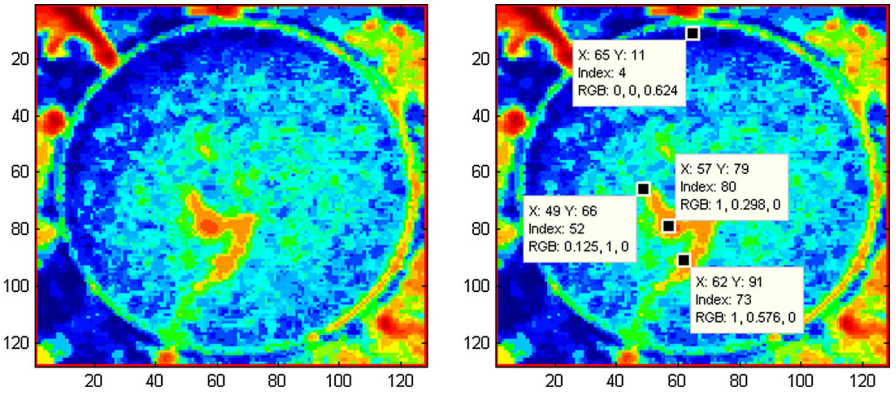

Fig. 6 Segmented output image for Seq1 with segment mean values in pixels. *To the right*, four selected segments in sequence

this as calculation time function plots. The FCM algorithm is computationally much more complex than KM; it is difficult to converge and requires choosing an appropriate fuzziness factor r for every change in the number of processed frames of the sequence. In our experiment, we adopted the constant $r = 1.5$ and the number of iterations is equal to 200. It is worthwhile to mention that the C# code does not produce a native code, and therefore is slower than an application written in C++, but it is relatively easy in a C# environment to implement parallel processing using a

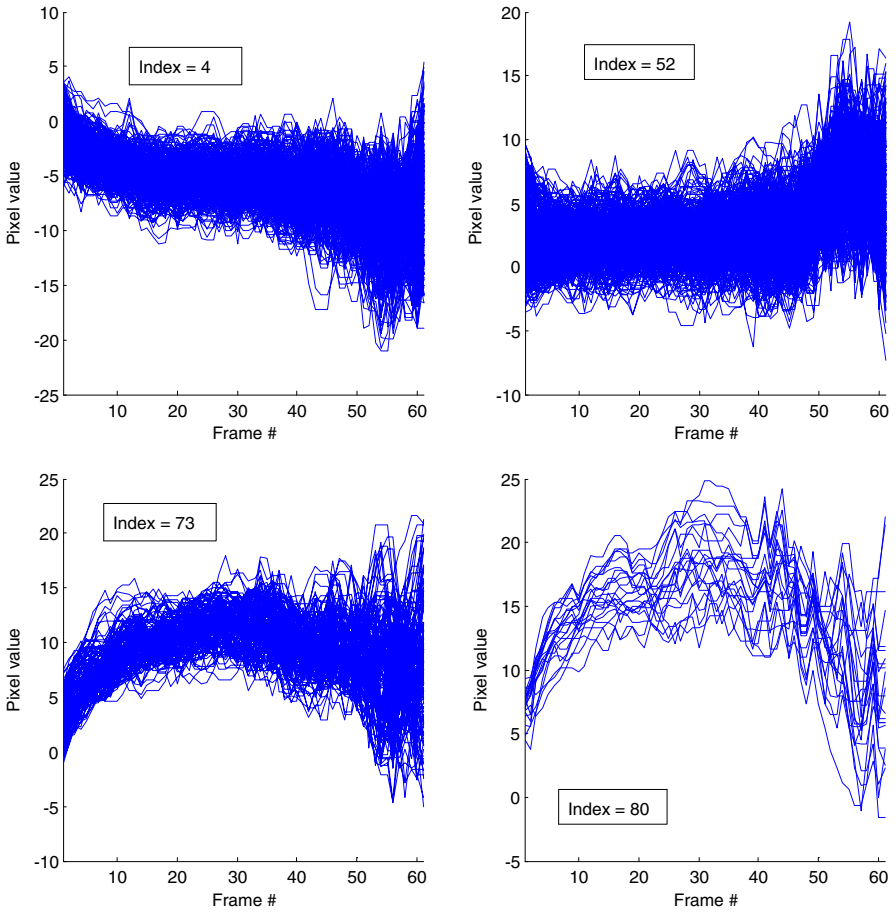

Fig. 7 All pixel characteristics for selected segments of Seq1

parallel. For loop, the computation time was therefore shorter than the MATLAB implementation by a factor proportional to the number of processor cores—in our case, twice. Dynamic update segmentation converges faster than other algorithms. The computation time is longer only because it does not use parallel processing of pixels. Changing the order of pixel processing would affect, in this case, the final result of segmentation.

Figure 6 presents a result image of Seq1 sequence segmentation in which pixel values are equal to the mean value of the associated segment's center vector. Using a jet palette for pseudocolors, the correctness of segmentation could be easily verified. Four segments from this image are chosen, and all pixel characteristics for a given segment are presented in images in Fig. 7.

All algorithms implemented in C# are object-oriented, enabling easy analysis and code modification. Implementation of other segmentation algorithms, such as hierarchical segmentation, is planned in the future.

4 Conclusion

This paper presents a new method for initializing a K-means algorithm, which takes into account the density of pixels in a sequence. A quick estimate method for calculating the density of pixels with a complexity of O(n) has also been presented. Using this additional information, the HD method obtains smaller segmentation errors than KKZ and FA initializations.

K-means segmentation using the kd-tree has proved to be very fast with segmentation of single images, but segmentation of the sequences takes much longer and successive iterations are not performed as fast as in standard implementations of a K-means algorithm. With sorting of one-dimensional data, a K-means algorithm using the kd-tree can be used to segment individual images in real time. Due to the recursive nature of the algorithm, there are considerable difficulties in accelerating consecutive iterations of the algorithm and parallelization of computations.

Sequences of thermal images are contained in each pixel of the frame, temperature information associated with the same location on the sample surface but also with the depth information associated with the frame number. This example of specific thermal image sequences enables very high information compression using cluster segmentation. As a result, pixels with similar thermal properties are combined into groups—clusters. Such a reduction in quantity of information can allow real-time implementation of thermal tomography.

References

1. E. Forgy, Biometrics **21**, 768 (1965)
2. J. MacQueen, in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1 (University of California Press, 1967), p. 281
3. S.P. Lloyd, IEEE Trans. Inf. Theory. **28**, 129 (1982)
4. X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A.F.M. Ng, B. Liu, P.S. Yu, Z. Zhou, M. Steinbach, D.J. Hand, D. Steinberg, Knowl. Inf. Syst. **14**, 1 (2008)
5. J.M. Pena, J.A. Lozano, P. Larranaga, Pattern Recognit. Lett. **20**, 1027 (1999)
6. I. Katsavounidis, C. Kuo, Z. Zhang, IEEE Signal Process. Lett. **1**, 144 (1994)
7. D. Arthur, S. Vassilvitskii, in *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms* (2007), p. 1027
8. S.J. Redmond, C. Heneghan, Pattern Recognit. Lett. **28**, 965 (2007)
9. J.C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms* (Plenum Press, New York, 1981)