Machine Vision and Applications
DOI 10.1007/s00138-016-0788-0

ORIGINAL PAPER

Motion estimation using learning automata

Bahman Damerchilu[1] · **Mohammad Sadegh Norouzzadeh**[2] · **Mohammad Reza Meybodi**[1]

Received: 22 July 2015 / Accepted: 7 June 2016

Abstract Block-matching algorithms (BMAs) are widely employed for motion estimation. BMAs divide input frames into several blocks and minimize an error function for each block to calculate motion vectors. Afterward, each motion vector is applicable for all of the pixels within the block. Since computing the error functions is resource intensive, many fast-search motion estimation algorithms have been suggested to reduce the computational cost. These fast algorithms provide a significant reduction in computation but often converge to a local minimum. A learning automaton is an adaptive decision-making unit that learns the optimal action through repeated interactions with its environment. Learning automata (LA) have been applied successfully to a wide range of applications including pattern recognition, dynamic channel assignment, and social network analysis. In this paper, we apply LA to motion estimation problem, which is one of the basic problems in computer vision. We compare the accuracy and performance of the suggested algorithms with other well-known BMAs. Interestingly, the obtained results indicate high efficiency and accuracy of the proposed methods. The results suggest that simplicity, efficiency, parallel nature, and accuracy of LA-based methods make them a good candidate to solve computer vision problems.

✉ Mohammad Sadegh Norouzzadeh
mnorouzz@uwyo.edu

Bahman Damerchilu
b.damerchilu@aut.ac.ir

Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

[1] Department of Computer, Amirkabir University of Technology, Tehran, Iran

[2] Department of Computer Science, University of Wyoming, Laramie, WY, USA

Keywords Motion estimation · Block-matching · Learning automata · Pursuit learning algorithm

1 Introduction

Understanding object movement patterns in a video has many applications in different disciplines such as transmission [1], storage [2], and management [3] of videos. Motion estimation is the process of determining motion vectors (MVs) which describe the transformation from one 2D image to another. This process can be applied for all successive frames of a video. In other words, MVs indicate the displacement of pixels in the current frame from the previous one [4]. The latest applications of the motion estimation include gesture recognition [5], hand posture analysis [6], breathing analysis [7], and video compression (e.g., MPEG4 and H.264) [8].

In a video sequence, there is a high level of similarity (i.e., redundancy) between consecutive frames. The idea of temporal redundancy reduction is to encode the reference frame and then encode only the differences between the reference frame and the consecutive ones. These differences are called "error" or "residual." Video coding techniques predict the content of the current frame based on the reference frame and these residuals. This process is known as motion compensation.

Although there are several different approaches for motion estimation, block-matching is the most popular approach due to its simplicity. In block-matching, a motion vector for an entire block of pixels is computed and then it is applied to all the pixels within the block. This method reduces computational cost and achieves more accurate motion vectors because typically objects consist of a cluster of neighboring pixels.

Published online: 20 June 2016

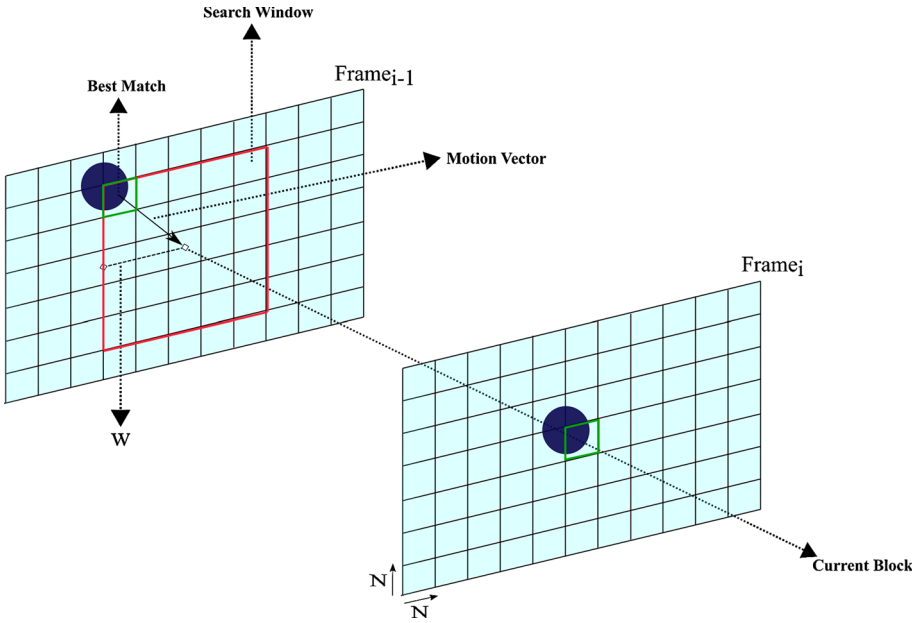

Fig. 1 In block-matching procedure, a fixed-size window in the previous frame is searched to find the best match for the current block. The vector that maps the best found match to the current block is called motion vector

In block-matching algorithms, each frame is separated into several non-overlapping blocks with the same size. For each block in the current frame, the best match is obtained within a search window of size $(2*w+1)*(2*w+1)$ of the previous frame, where w is the maximum allowed displacement in both horizontal and vertical directions. Afterward, the motion vector for the current block is the position difference between the current block and its best-matched block in the previous frame (Fig. 1).

The underlying assumption behind BM is the existence of a strong correlation between each pixel and its neighbors. Therefore, a motion vector can be assigned to a block of pixels instead of individual pixels.

In block-matching approach, motion vectors are computed by minimizing error functions, which are representing the grayscale difference between a block in the current frame and its match in the previous frame. Therefore, the best match for the current block will have the lowest error function among all candidate blocks within a search window [9,10].

Different evaluation criteria have been suggested to calculate the distance between two blocks such as mean of absolute difference (MAD), sum of absolute difference (SAD), and mean square error (MSE). The most commonly used metric is SAD, which is defined in Eq. (1).

$$\mathrm{SAD}_{(u,v)} = \sum_{i=1}^{N}\sum_{j=1}^{N} |f_t(i,j) - f_{t-1}(i+u, j+v)| \qquad (1)$$
$$-w \le u, v \le w$$

In Eq. (1), (u, v) is the distance of the candidate block to the current block, $f_t(i, j)$ implies the graylevel value of (i, j)th pixel in the current frame, and $f_{t-1}(i, j)$ shows the graylevel value of (i, j)th pixel in the previous frame. The displacement vector (u, v) of the candidate block with the minimum $SAD_{(u,v)}$ is the motion vector. The size of block is $N * N$ and u, v are in the range $[-w \quad w]$.

The full search algorithm (FSA) is known as the most robust and accurate method for finding the motion vectors. In order to find the block with the minimum SAD, FSA tests all the possible candidate blocks within the search window. Even though this algorithm is the most computationally expensive choice, it is able to find the best match. Efficiency and accuracy of the other block-matching methods are measured by comparing them to FSA.

Block-matching algorithms evaluate some matching criteria (MC) to compute the best match for each block. The calculation of MC is generally expensive; therefore, several algorithms have been proposed to speed it up. These algorithms can be classified into the following categories:

1. *Using a fix search pattern*: This category involves methods with a predefined pattern to search for the best match. In these methods, the search is performed on a fixed subset of the search window. This category contains three-step search (TSS) [11], new three-step search (NTSS) [12], simple and efficient TSS (SES) [13], four-step search (4SS) [14], and diamond search (DS) [15].
2. *Reducing the number of search points*: In these algorithms, search points are chosen in an iterative way to minimize the error functions. These algorithms include adaptive rood pattern search (ARPS) [16], block-based gradient descent search (BBGD) [17], neighborhood elimination algorithm (NE) [18], block-matching algorithm based on differential evolution (DE-BM) [19], and

block-matching algorithm based on harmony search optimization (HS-BM) [19].

3. *Reducing the computational cost of error functions*: Less computationally expensive matching criteria are used in this type of algorithms. New pixel-decimation (ND) [20], successive elimination algorithms [21], and winner-update strategy [22] are three examples of this category.

In this paper, we propose three novel algorithms based on learning automata for motion estimation. Our proposed algorithms can be classified as the second category. Learning automaton is an abstract model which learns to choose the best action among its set of actions. This model has been successfully applied to solve different sorts of engineering problems such as pattern recognition [23], image segmentation [24], priority assignment [25], and social network analysis [26]. In this paper, we employ LA for motion estimation, which is one of the basic problems in machine vision.

2 Learning automata

Learning automaton is an adaptive decision-making unit that learns the best action from its set of actions through repeated interactions with a random environment (Fig. 2). Learning automaton maintains a probability vector, which contains the probability of choosing each action at any time instant. The learning process is as follows: the automaton randomly chooses one of its actions based on its probability vector. The selected action is rewarded or penalized by the environment as a response. The automaton takes the response into account and modifies its probability vector. The main goal of such automaton is to determine the optimal action from a set of actions, where the optimal action is defined as one maximizing the expected reward [27–29].

In general, a learning automaton is defined by (A, Q, R, T) and the environment is defined by (A, R, D), where $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of all actions of the automaton, where α_i implies the automaton output and the environment input. Also, R is the domain of responses from the environment and $\beta(k) \in R$ is the response signal that the automaton receives from the environment at kth time instant. In addition, $D = \{d_1, d_2, \ldots, d_r\}$ is the set of reward probabilities, where $d_i = E\{\beta(k) \mid \alpha(k) = \alpha_i\}$. In other words, d_i variable demonstrates the average reward given to action α_i. If d_i is independent of k, environment is called stationary; otherwise, it is non-stationary. Usually the set of reward probabilities is unknown for the automaton; therefore, an estimate for these set is used. Moreover, Q is the state of the automaton which is defined by $Q(k) = \left[P(k), \hat{D}(k)\right]$ where $P(k) = \{P_1(k), P_2(k), \ldots, P_r(k)\}$, $\sum_{i=1}^{r} P_i(k) = 1$ is the action probability vector, and $\hat{D}(k) = \left\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_r\right\}$ is the estimation of the reward probabilities at the kth time instant. T is the learning algorithm which is used by the automaton for updating its state.

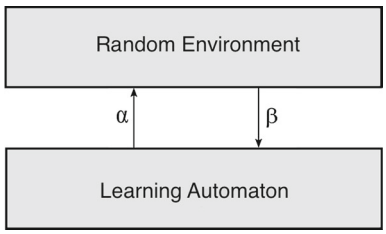

Fig. 2 Learning automaton learns the best action through repeated interaction with a random environment. Automaton performs an action α on the environment and the environment responds with signal β

There exist a variety of different learning automata for handling various learning problems. Some examples of these varieties are finite action set learning automata (FALA) [29], parameterized learning automata (PLA) [29], generalized learning automata (GLA) [30], continuous action set learning automata (CALA) [29], game of LA [29], and network of LA [31]. Here, we explain the learning automata that we used in details.

2.1 Pursuit learning automata

In order to design a learning automata algorithm with a high convergence rate, Thathachar and Sastry designed a new class of learning algorithms called estimator algorithms [32]. These algorithms preserve the current estimates for the reward probability of each action and utilize them for updating the action probabilities. Both long- and short-term properties of the environment can be considered by aforementioned algorithms. Long-term properties include the current reward probability estimates, while short-term properties comprise the current environment response.

Pursuit learning automata are a subset of the estimator algorithms, which only pursues the current estimated optimal action. This algorithm is firstly presented by Thathachar and Sastry [32]; initially, only the long-term properties of the environment were used. In this algorithm, action set is considered continuously. Oommen and Lanctôt [33] proposed a discretized version of the pursuit algorithm which considers both the short-term and the long-term properties of the environment. The only difference between the discrete and the continuous versions of pursuit algorithms is the updating rule for the action probabilities. The discrete pursuit algorithms make changes to the probability vector $P(k)$ in discrete steps, while the continuous ones use a continuous function to update $P(k)$.

Pursuit learning automata have two more vectors $W(k) = \{W_1(k), W_2(k), \ldots, W_r(k)\}$ and $Z(k) = \left\{Z_1(k), Z_2(k), \ldots,\right.$

$Z_r(k)\Big\}$ for calculating the estimates of the reward probabilities $\hat{D}(k) = \left\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_r\right\}$. $W_i(k)$ indicates the number of time that ith action has been rewarded and $Z_i(k)$ is the number of times that ith action has been selected up to time k. Therefore, $\hat{D}(k)$ defined Eq. (2).

$$\hat{d}_i(k) = \frac{W_i(k)}{Z_i(k)}, \quad 1 \le i \le r \tag{2}$$

The discrete pursuit algorithm [33] involves three steps. In the first step, each automaton chooses action $\alpha_i(k)$ based on probability vector $P(k)$ and receives response $\beta(k)$ from the environment. The second step encompasses updating probability distribution $P(k)$ as E1. 3.

$$\begin{aligned} p_j(k+1) &= \max\{p_j(k) - \Delta, 0\} \qquad 1 \le j \le r \ , \ j \ne m \\ p_m(k+1) &= 1 - \sum_{j \ne m} p_j(k+1) \end{aligned} \tag{3}$$

In Eq. (3), m is the index of the maximal component of reward probability estimate vector, so $\hat{d}_m = \max\{\hat{d}_j\}$. $\Delta = 1/(r * n)$ exhibits the step size, and n is the resolution parameter. Therefore, in this step, probabilities of all the actions that do not correspond to the highest estimate of reward probabilities are decreased and the probability of the action with the highest estimate is increased.

Third step of this algorithm consists of updating the reward probability estimate vector as Eq. (4) where i is the index of the selected action in first step.

$$\begin{aligned} W_i(k+1) &= W_i(k) + (1 - \beta(k)) \\ Z_i(k+1) &= Z_i(k) + 1 \\ \hat{d}_i(k+1) &= \frac{W_i(k+1)}{Z_i(k+1)} \end{aligned} \tag{4}$$

2.2 Team of learning automata

One of the possible configurations for learning automata is the game of automata [29]. This configuration consists of N distinct automaton, where each of them is trying to find the optimal action from its own action set.

Let $A^1, A^2, \ldots, A^N$ be the N automaton involved in the game and $P^1(k), P^2(k), \ldots, P^N(k)$ are the action probability distribution of the N automata. At each time instant k, each automaton A^i chooses an action $\alpha^i(k)$ independently and randomly according to the probability distribution of $P^i(k)$. This set of N actions will be given to the environment and the environment responds with N random rewards. Let $\beta^i(k)$ be the response of the environment to automaton A^i, for choice of action at time instant k. Automaton A^i uses $\beta^i(k)$ and updates $P^i(k)$ by a learning algorithm.

Having a common reward for all the automata is one of the interesting features of this model. In other words, all the automata get the same reward from the environment; therefore, $\beta^1(k) = \beta^2(k) = \cdots = \beta^N(k) = \beta(k)$. This condition of the game is often referred to as a team of learning automata. A team of automata is usually useful for optimizing multivariate functions; each automaton in the team is trying to identify the optimum value of a particular parameter. In this case, the action set of each automaton includes the possible values of the corresponding parameters.

2.3 Variable action set learning automata

When the number of actions of a learning automaton varies with time, it is called variable action set learning automaton [34]. Assume $V(k) \subset A$ is a non-empty subset of the actions at time k, $V(k)$ is called active action set, which represents the available actions of the learning automaton. In this type of learning automaton, in each iteration of the algorithm, automaton selects one action from the active action set according to the scaled probability vector $\widehat{P}(k)$ which is defined as Eq. (5) where $N(k)$ is sum of the probabilities of the actions in subset $V(k)$.

$$\begin{cases} \widehat{P^i}(k) = \frac{P_i(k)}{N(k)} & \alpha^i \in V(k) \\ \widehat{P^i}(k) = 0 & \alpha^i \notin V(k) \end{cases} \tag{5}$$

Depending on the response received from the environment, the automaton updates its scaled action probability vector $\widehat{P}(k)$. It must be noted that we only update the probabilities of the active actions. Finally, the probability vector of the actions of the chosen subset is rescaled (Eq. 6).

$$\forall \alpha_i \in V(k) : P^i(k+1) = \widehat{P^i}(k+1) * N(k) \tag{6}$$

3 Motion estimation using learning automata

As mentioned earlier, block-matching algorithms aim to identify the best match for the current block within a search window of the previous frame. If we consider candidate blocks in the search window as actions of automata, then the process of identifying the best matches can be performed by learning automata.

In this section, we explain three algorithms based on learning automata for motion estimation.

3.1 Motion estimation using pursuit learning automata

Description of this method is as follows: one learning automaton is assigned to each block of the current frame

and the candidate blocks in the search window are considered as actions of the automaton. In each iteration of the algorithm, each automaton selects an action (α_i) according to its probability vector. Error criterion (SAD) between the current block and the block corresponding to the selected action is calculated and normalized to interval [0 1] and is given to the automaton as the response of the environment. If α_i is selected again in the next iterations of the algorithm, the error criterion is not calculated again and the stored value in memory will be used. In the next step of the algorithm, the automaton updates its action probability vector according to the Eq. (3). In the last step, the automaton updates the reward probability estimate vector using the received response from the environment (Eq. 4). Finally, the action with the highest probability is selected as the best match for the current block. The Algorithm 1 shows the pseudocode of pursuit learning automata-based motion estimation.

One of the parameters which affect the performance of the learning automaton is the number of actions. In other words, having a large number of actions reduces the efficiency of learning automaton. Assume $w = 7$ therefore the number of actions is equal to 225 (i.e., $(2w + 1) * (2w + 1)$), and this number of actions is not suitable for automata. In this study, we propose two approaches to deal with this issue in Sects. 3.2 and 3.3.

3.2 Motion estimation using variable action set learning automata

The successive elimination algorithm (SEA) proposed in [21] uses an upper bound to eliminate the impossible candidate blocks for reducing the computation of motion estimation. This algorithm adopts the well-known Minkowski inequality concept to derive the following inequality (Eq. 7) where X and Y are two blocks.

$$S(X,Y) = \left| \sum_{i=1}^{N}\sum_{j=1}^{N} X(i,j) - \sum_{i=1}^{N}\sum_{j=1}^{N} Y(i,j) \right|$$
$$\mathrm{SAD}(X,Y) \geq S(X,Y) \tag{7}$$

In SEA, if the difference of block-sums (i.e., sum of the pixels within a block) between the current block and the candidate block is greater than the upper bound, then it is impossible for this candidate block to be the best match block. The block-sum pyramid algorithm (BSPA) [35] made an extension of Eq. (7) to a multi-resolutional pyramid form. In BSPA, a pyramid-like hierarchy for the current block and the candidate blocks is constructed. Each pixel in the m-th level is the sum of $2 * 2$ neighboring pixels in the $(m-1)$-th level (Fig. 3), which is defined as Eq. (8).

$$\begin{aligned} X^m(i,j) = {} & X^{m-1}(2*i-1, 2*j-1) \\ & + X^{m-1}(2*i-1, 2*j) \\ & + X^{m-1}(2*i, 2*j-1) \\ & + X^{m-1}(2*i, 2*j) \end{aligned} \tag{8}$$

Algorithm 1 Pursuit learning automata based motion estimation (PLA-ME)

Parameters:

m	Index of the maximal component of the reward estimate vector.
$W_i(t)$	Number of times that ith action has been rewarded up to time t
$Z_i(t)$	Number of times that ith action has been selected up to time t
N	Block size
w	Search windows size
$r = (2 * w + 1)^2$	Number of actions
n	Resolution parameter
$\Delta = 1/(r * n)$	Smallest step size
$\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$	Actions(Candidate blocks)
$P = \{p_1, p_2, ..., p_r\}$	Action probability vector
SAD_i	Sum of absolute difference for i-th action. $\forall i \;\; SAD_i = NULL$

Method
Repeat

Step1 At time t pick $\alpha(t)$ according to probability vector $P(t)$

Step2
if $SAD_i = NULL$ **then**
calculate SAD value for ith candidate block and normalize it to interval [0 1]
end
$\beta(t) = SAD_i$

Step3 Update $P(t)$ according to following Eq.:
$p_j(t+1) = max_{j \neq m}\{p_j(t) - \Delta, 0\}$
$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1)$

Step4 Update $\widehat{d}(t)$ according to following Eq.:
$W_i(t+1) = W_i(t) + (1 - \beta(t))$
$Z_i(t+1) = Z_i(t) + 1$
$\widehat{d_i}(t+1) = W_i(t+1)/Z_i(t+1)$

End Repeat

best match = α_i, $p_i = max_j\{p_j\}$

End Method

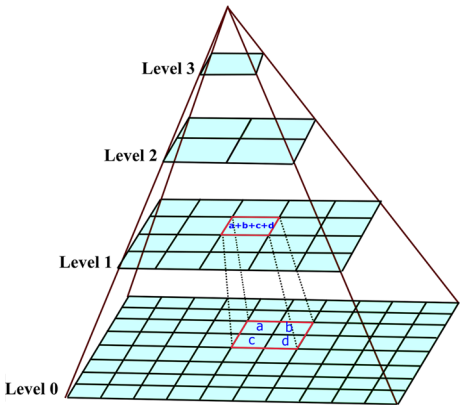

Fig. 3 Constructing a block-sum hierarchy pyramid can help the algorithm to search fewer points. In block-sum hierarchy pyramid, each pixel in the m-th level is the sum of $2 * 2$ neighboring pixels in the $(m-1)$-th level

For an $N * N$ block , the m-th level SAD is defined as Eq. (9).

$$\mathrm{SAD}^m(X,Y) \sum_{i=1}^{2^{M-m}} \sum_{j=1}^{2^{M-m}} \left| X^m(i,j) - Y^m(i,j) \right| \tag{9}$$

Then the following multi-resolutional Minkowski inequality can be obtained [35] (Eq. 10).

$$\begin{aligned} &\mathrm{SAD}^0(X,Y) \geq \mathrm{SAD}^1(X,Y) \geq \cdots \geq \\ &\quad \mathrm{SAD}^{M-1}(X,Y) \geq \mathrm{SAD}^M(X,Y) \end{aligned} \tag{10}$$

Equation (10) can be used as a measure for determining and eliminating useless actions in pursuit learning automata-based motion estimation. Doing so, the number of actions for each automaton will be reduced during the learning process and consequently the efficiency of the learning automata will be increased.

In order to implement this strategy, we use variable action set learning automata model. Description of this algorithm is as follows: first, block-sum pyramid for the current block (T) is constructed. At each time instant, the automaton chooses action α_i randomly based on its probability vector. If α_i is selected for first time, the block-sum pyramid for candidate block (C) corresponding to α_i is constructed. Then the SAD value of the top level of the pyramid will be checked. If $\mathrm{SAD}^M(T,C) > \mathrm{SAD}_{\min}$, where $\mathrm{SAD}_{\min}$ is the current achieved minimum SAD value, this action can be eliminated from the action set of the automaton. Otherwise, SAD value on next level of pyramid will be checked. If $\mathrm{SAD}^{M-1}(T,C) > \mathrm{SAD}_{rmin}$, for the same reason, this action can be eliminated from the action set of the automaton as well. This process is repeated until this action is eliminated or the bottom level of the pyramid is reached. At the bottem level of the pyramid if $\mathrm{SAD}^0(T,C) > \mathrm{SAD}_{\min}$, value of $\mathrm{SAD}_{\min}$ is replaced with $\mathrm{SAD}^0(T,C)$. In this level, $\mathrm{SAD}^0(T,C)$ is normalized to interval [0 1] and will be given to the automaton as the response of the environment. Automaton uses this response and updates the action probability distribution according to the pursuit learning paradigm. Algorithm 2 shows the pseudocode of variable action set learning automata-based motion estimation (VASLA-ME).

Algorithm 2 Variable action set learning automata based motion estimation (VASLA-ME)

Parameters:

	Same as parameter that defined in PLA-ME
$SAD_{min}(t)$	Minimum achived SAD up to time t
T	Template block

Method
Repeat

Step1 At time t pick $\alpha(t)$ according to probability vector $P(t)$

Step2
if $SAD_i = NULL$ **then**
construct block-sum pyramid for ith candidate block(C)
for $m = \log N \; to \; 1$ **do**
if $SAD^m(T,C) \geq SAD_{min}(t)$ **then**
Remove α_i
Update probabily vector as **Step3**
Go to **Step1**
end
end
$SAD_i = SAD^0(T,C)$
end
$\beta(t) = SAD_i$

Step3 Update $P(t)$ exactly as in the PLA-ME

Step4 Update $\widehat{d}(t)$ exactly as in the PLA-ME

End Repeat

best match = α_i, $p_i = max_j\{p_j\}$

End Method

3.3 Motion estimation using team of pursuit learning automata

Each candidate block in the search window has a certain amount of vertical (dx) and horizontal (dy) displacement relative to the current block, where $-w \leq \mathrm{d}x, \mathrm{d}y \leq w$.

Table 1 Test sequences used in the comparison test

Sequence	Format	Total frames	Motion type
Container	$QCIF$(176*144)	299	Low
Akiyo	$QCIF$(176*144)	300	Medium
Carphone	$QCIF$(176*144)	381	Medium
Foreman	$QCIF$(352*288)	398	Medium
Stefan	CIF (352*288)	89	High
Tennis	SIF(352*240)	150	High

According to this fact, a team with two learning automata can be used for motion estimation. In this team, one of the automata determines the optimal displacement value along the horizontal direction and the other one is responsible for specifying the optimal displacement value along the vertical direction. In this case, the number of actions (r) for each of the learning automata of the team is equal to $2 * w + 1$. Assuming $w = 7$, the number of actions for each learning automaton is equal to 15, which is suitable for automata.

We used two pursuit learning automata (PLA) in the team. Each of PLA^i, $i \in \{1, 2\}$ in the team has $r = 2 * w + 1$ actions and $\alpha^i = \{\alpha_1^i, \alpha_2^i, \ldots, \alpha_r^i\}$ is the action set of PLA^i, where for $i = 1$ this set is equivalent to the horizontal displacement value of the current block in the search window and for $i = 2$ is equivalent to the vertical displacement value of the current block within the search window. Thus, $\alpha^1 \times \alpha^2$ is the set of candidate blocks in the search window. $P^i = \{p_1^i, p_2^i, \ldots, p_r^i\}$ is the action probability distribution of PLA^i.

Description of team of pursuit learning automata-based motion estimation (TPLA-ME) is as follows: at time instant k, each PLA^i chooses one of its actions randomly according to its action probability vector P^i. Assume α_i^1 is selected by PLA^1 and α_j^2 is selected by PLA^2; therefore, $(i - w - 1, j - w - 1)$ is the motion vector of the selected candidate block in search window. If this candidate block has been selected for the first time, error criteria (SAD) between the current block and the selected candidate block is computed and normalized to interval [0 1] and stored in memory. This value is given to each PLA^i as the response of the environment. Each PLA^i uses this response and updates its action probability vector independently. In the next iteration of the algorithm, if this candidate block is selected again, SAD value will not be calculated again and the stored value in memory will be used.

Finally, the candidate block with $(i - w - 1, j - w - 1)$ motion vector is selected as the best match where $p_i^1 = \max_{1 \le n \le r}\{p_n^1\}$ and $p_i^2 = \max_{1 \le n \le r}\{p_n^2\}$.

In this algorithm, we also use a reward approximation method based on nearest neighbor interpolation (NNI), which is proposed in [19]. According to this method, we estimate the reward of an action based on neighboring actions instead of calculating the matching criterion. If an action (α_s) is selected for the first time by the learning automaton, the environment calculates the reward of this action as follows: the nearest action (α_n) to the selected action is determined from the previously selected actions by LA. If the distance between α_s and α_n is less than d and matching criterion of α_n (SAD$_{\alpha_s}$) does not correspond to the best value, SAD$_{\alpha_n}$ will be used as the reward of α_s; otherwise, matching criterion of α_s will be calculated by Eq. (1).

3.4 The initial value of action probabilities

The first step of every LA algorithm is to generate an initial action probability vector. Typically these values are initialized uniformly, by assuming lack of knowledge about the problem. On the other hand, in some applications, action probability vector can be weighted more specifically through some domain knowledge, which can significantly improve the performance of LA. To obtain an appropriate initial action probability vector, we performed an analysis over the distribution of motion vectors. For this purpose, we calculate the motion vector probability distributions of several video sequences (see Table 1) using FSA with SAD as the matching criterion and $w = 7$. The distribution of motion vectors for Foremen and Stefan sequences are shown in Fig. 4.

According to these analyses, the most video sequences contain center-biased motion vector distributions. It could be noted that 98 % of the motion vectors lie at the origin of the search window for a slow moving sequence, such as

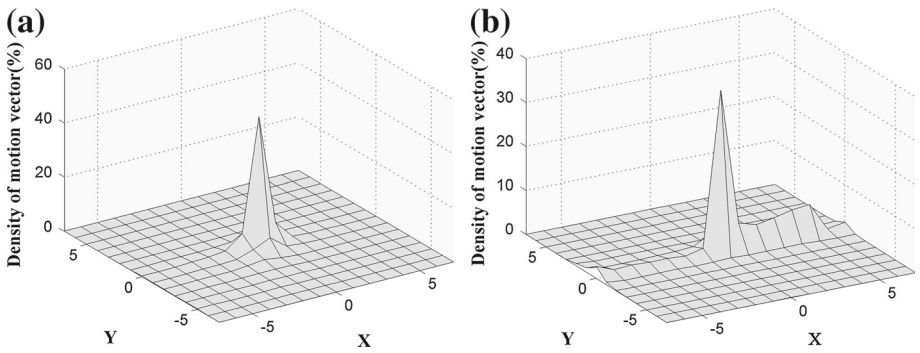

Fig. 4 By analyzing the distribution of motion vectors, we can choose a more meaningful initial start point for our algorithm. Average motion vector distribution over a search window of size $w = 7$ for Foreman (**a**) and Stefan (**b**) datasets

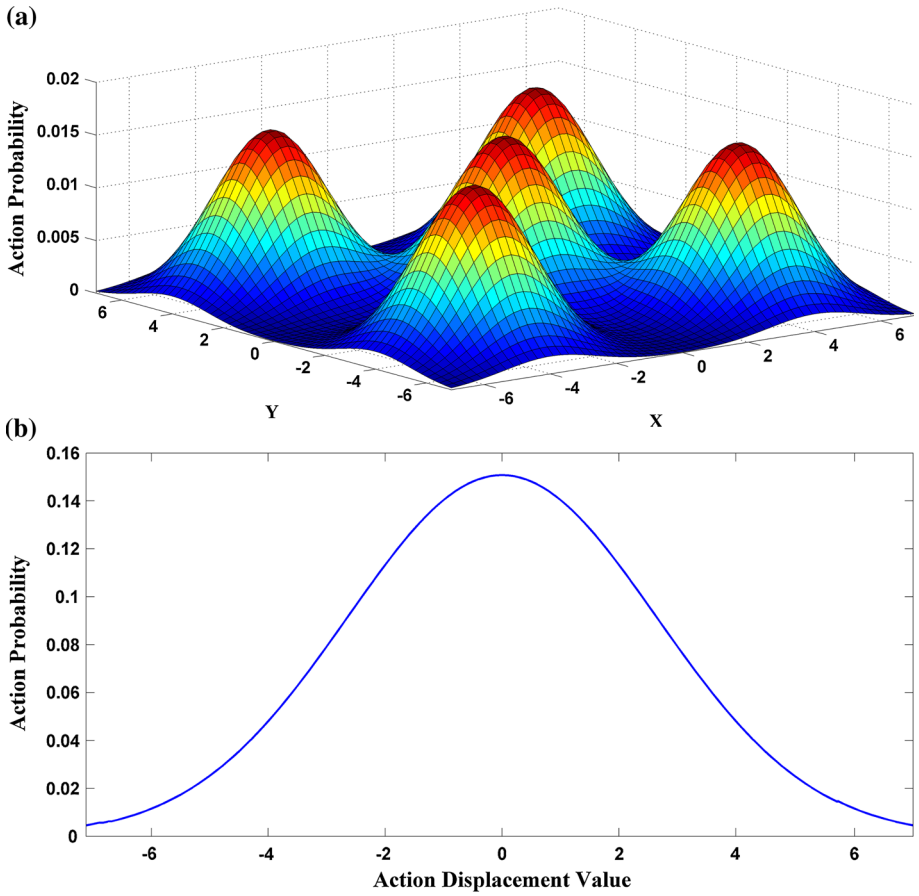

Fig. 5 Choosing a good initial action probability distribution can effectively help learning automata to have a better accuracy and performance. The initial action probability distribution of PLA-ME and VASLA-ME (**a**), and TPLA-ME (**b**)

Container, while complex motion sequences, such as Carphone and Foreman, have only 53.5 and 46.7 % of their motions in center of the search window, respectively. The Stefan sequence, showing the most complex motion content, was found to have only 36.9 % of the motions in center.

The initial action probability distribution for PLA-ME and VASLA-ME is considered as a mixture of 2d-Gaussians with five center points located at $[0\ 0]$, $[w/2\ \ w/2]$, $[-w/2\ \ w/2]$, $[w/2\ \ -w/2]$ and $[-w/2\ \ -w/2]$. Figure 5a shows the surface of this action probability distribution. For TPLA-ME, we use a 1d-Gaussians distribution for each of automata in the team. This distribution is shown in Fig. 5b.

3.5 Efficient implementation of pursuit learning algorithm

In the three proposed algorithms (PLA-ME, VASLA-ME, and TPLA-ME), we use discretized pursuit learning paradigm [33]. In these three algorithms, the environment is stationary and $\hat{d}_i$ is invariant under time instant k, therefore we can remove vector Z and W from Eq. (4) in the pursuit learning algorithm which is defined $\hat{d}_i$ as Eq. (11) where β_i is the SAD value between current block and candidate block corresponding to the selected action α_i.

$$\hat{d}_i = 1 - \beta_i \tag{11}$$

By considering these changes, the proposed idea in [36] can be utilized for efficient implementation of the discretized pursuit algorithm. Based on this idea, we use a variable size array F in which each cell of this array belongs to a specific action. Each considered cell in this array represents a probability equal to Δ. In order to select an action, we select one of the cells randomly and then the corresponding action to this cell is chosen. Therefore, $p_i = \frac{N_i}{\text{length}(F)}$, where N_i is the number of cells corresponding to α_i in array F. In order to update the action probability, in each time instant, one of the cells corresponds to all actions (except α_i) and is assigned to α_i, where $\hat{d}_m = \max_{1 \le j \le r}\{\hat{d}_j\}$. Figure 6 illustrates an example of this update scheme. In this example, a DPLA with $r = 4$ and $n = 3$ is considered and each action

Fig. 6 DPLA with $r = 4$ and $n = 3$, **a** initialization, **b** update action probability when α_1 selected as action with highest estimate of reward probabilities

Table 2 PSNR and D_{PSNR} value comparison for all motion estimation methods

Algorithm	Container		Carphone		Akiyo		Foreman		Stefan		Tennis		Average of D_{PSNR}
	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	
FSA	**43.18**	**0**	**31.51**	**0**	**29.07**	**0**	**31.69**	**0**	**25.95**	**0**	**35.74**	**0**	**0**
TSS	43.1	0.2	30.27	3.92	26.21	9.84	29.37	7.32	21.14	18.52	30.58	14.43	9.03
4SS	43.12	0.15	30.24	4.01	26.21	9.84	29.34	7.44	21.41	17.48	30.62	14.32	8.84
NTSS	43.12	0.15	30.35	3.67	27.12	6.71	30.56	3.57	22.52	13.2	31.21	12.67	6.66
SESTSS	43.12	0.15	30.09	4.50	26.10	10.21	29.24	7.73	21.37	17.64	30.20	15.5	9.28
ARPS	43.08	0.23	30.23	4.06	26.12	10.14	29.46	7.03	21.45	17.34	29.95	16.20	9.16
BBGD	43.14	0.11	31.3	0.67	28.1	3.33	31	2.19	25.17	3.01	33.17	7.20	2.75
DS	43.13	0.13	31.26	0.79	28	3.7	31.19	1.59	24.98	3.73	33.98	4.92	2.47
NE	43.15	0.07	31.36	0.47	28.23	2.89	31.23	1.47	25.22	2.81	33.88	5.20	2.15
PSO-BM	43.15	0.07	31.39	0.38	28.33	2.55	31.27	1.34	25.39	2.15	33.91	5.12	1.93
DE-BM	43.17	0.04	31.47	0.13	28.78	1	31.51	0.58	25.15	3.09	34.77	2.71	1.26
HS-BM	43.16	0.05	31.40	0.34	29.01	0.21	31.55	0.45	25.05	3.46	33.54	6.15	1.77
PLA-ME	**43.18**	**0**	31.42	0.28	**29.07**	**0**	31.43	0.82	25.12	3.19	34.55	3.32	1.25
TPLA-ME	**43.18**	**0**	31.42	0.28	**29.07**	**0**	31.42	0.85	25.22	2.92	34.88	2.40	1.07
VASPLA-ME	**43.18**	**0**	**31.51**	**0**	**29.07**	**0**	**31.69**	**0**	**25.58**	**1.42**	**35.26**	**1.34**	**0.46**

has been presented by a different color. Therefore, each cell that is shown in Fig. 6 has a probability of $\Delta = 1/12$. More details of this implementation can be found in [36].

In this implementation, both action selection and update the action probability vector are performed without considering the number of actions and they do not require more operations.

4 Experiments

We perform several experiments over the popular video sequences listed in Table 1. These sequences consist of different degrees and types of motion, and they are in QCIF (176*144), CIF (352*288), and SIF(352*240) formats. Among these sequences, Container has low motion changes. Carphone, Foreman, and Akiyo have moderately complex motions. Complex motion content can be found in Stefan and Tennis sequences.

Each frame is partitioned into blocks with the size of $16 * 16 (N = 16)$ pixels for motion estimation. Thus, the maximum displacement within the search window is seven pixels in both horizontal and vertical directions.

To evaluate the performance of the proposed algorithms, we implement different block-matching algorithms, such as FSA, TSS [11], 4SS [14], NTSS [12], BBGD [17], DS [15], ARPS [16], SESTSS [13], NE [18], PSO-BM [37], DE-BM [19], and HS-BM [19]. All simulations were performed on a Intel Core i7 2.2 GHz computer with 8 GB of memory.

In our comparison, two relevant performance indicators are considered: the estimation quality and the search efficiency.

4.1 Estimation quality

First, all the algorithms are compared in terms of the distortion that they make which is characterized by the peak signal-to-noise ratio (PSNR) value. PSNR indicates the reconstruction quality when the motion vectors, computed through a motion estimation approach, are used for reconstructing the current frame. In PSNR, the signal is the original data frames whereas the noise is the error introduced by calculating the motion vectors. The PSNR defined Eq. (12) where MSE is the mean square error between the original frames and those estimated by the motion vectors and the reference frame.

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{255^2}{\text{MSE}}\right) \tag{12}$$

The PSNR degradation ratio (D_{PSNR}) is another performance index which is used in the comparison. This ratio expresses the level of mismatch between the PSNR of a motion estimation approach and full search algorithm (FSA). The D_{PSNR} value is in the ranges [0 100] and demonstrates high accuracy for the motion estimation approach when it is close to zero. The D_{PSNR} is defined in Eq. (13).

$$D_{\text{PSNR}} = \left(\frac{\text{PSNR}_{\text{FSA}} - \text{PSNR}_{\text{MB}}}{\text{PSNR}_{\text{FSA}}}\right) \times 100 \tag{13}$$

Table 2 shows the PSNR and D_{PSNR} values for different algorithms. In case of slow motion sequences, the PSNR and D_{PSNR} values of all algorithms are very similar. For the medium motion content sequences such as Carphone, Foreman, and Akiyo, the algorithms based on fixed patterns (TSS,

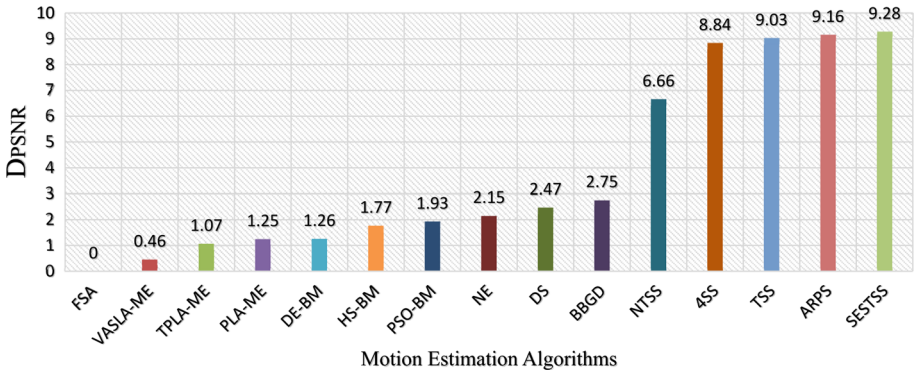

Fig. 7 Average D_{PSNR} obtained for all motion estimation algorithms over all sequences. All the three proposed algorithms achieved a good D_{PSNR} value in comparison with the other algorithms

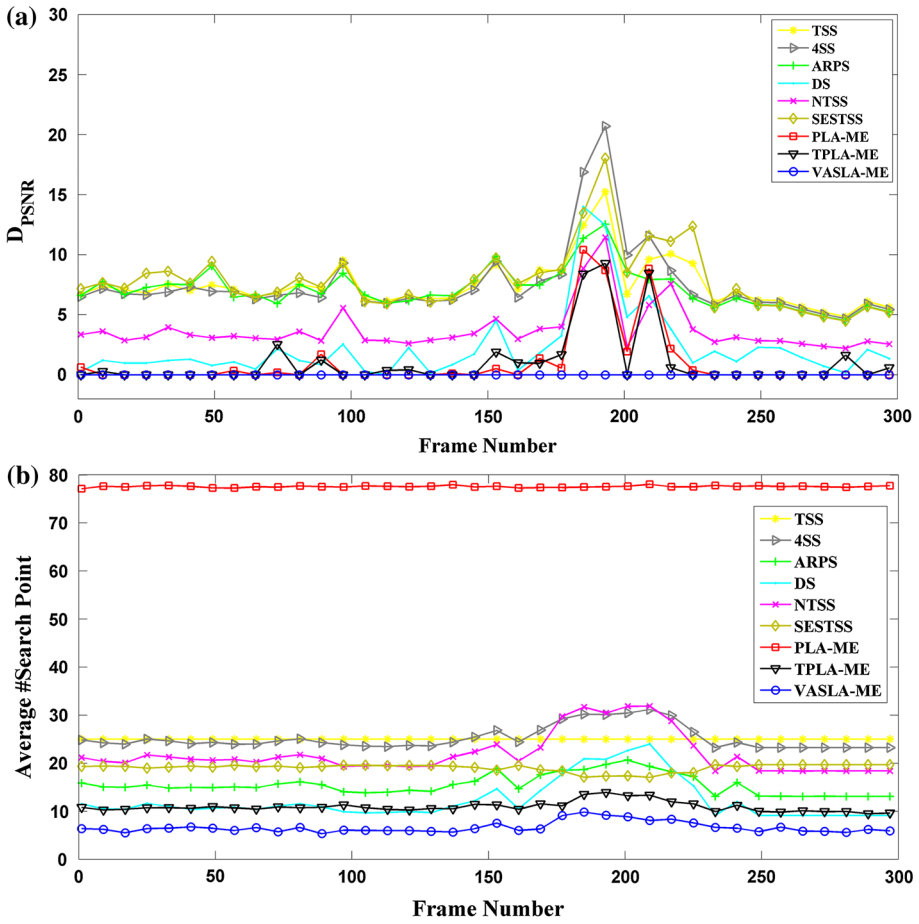

Fig. 8 Frame-wise performance comparison between different block-matching algorithms on "Foreman" sequence based on D_{PSNR} (**a**) and the mean number of search point (**b**). The proposed methods do a better job on finding the best match

NTSS, SESTSS, 4SS, and DS) exhibit the worst PSNR and high D_{PSNR} value and the algorithms based on function minimization (BBGD, ARPS and NE) have medium accuracy. The methods that use evolutionary algorithms (PSO-BM, DE-BM and HS-BM) are more accurate than the other two groups. On the other hand, our three proposed algorithms (PLA-ME, VASLA-ME and TPLA-ME) have the lowest D_{PSNR}, very close to FSA, which has been considered as the reference. For the high motion sequences of Stefan and Tennis, since the motion content of these sequences are complex, the performance, in general, becomes the worst for many of the algorithms, especially for those based on fixed patterns. However, in this case, the algorithm VASLA-ME and TPLA-ME have a better performance than the other motion estimation algorithms (Figs. 7, 8, 9).

As a summary of the estimation quality performance, Fig. 7 and the last column of Table 2 present the average D_{PSNR} obtained over all sequences. According to these values, the proposed algorithms in this work (PLA-ME, VASLA-ME, and TPLA-ME) are superior to the other approaches and VASLA-ME has the lowest D_{PSNR} value.

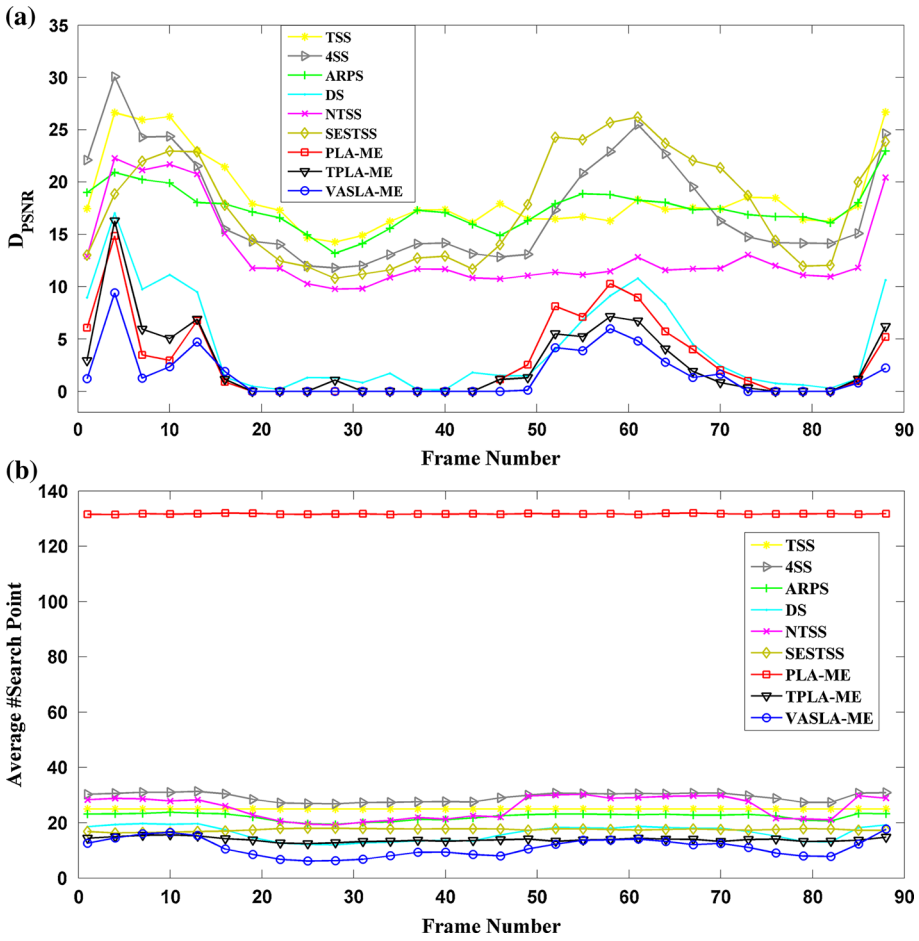

Fig. 9 Frame-wise performance comparison between different block-matching algorithms on "Stefan" sequence based on D_{PSNR} (**a**) and the mean number of search point (**b**). The proposed methods do a better job on finding the best match

We also compare the frame-wise performance of the proposed algorithms with traditional block-matching algorithms. For ease of readability, we compare the performance of the proposed algorithms only with a few representative block-matching algorithms.

We compare the frame-wise D_{PSNR} values of PLA-ME, VASLA-ME, and TPLA-ME with TSS, NTSS, SESTSS, 4SS, DS, ARPS for medium motion Foreman (Fig. 8a) and high motion Stefan (Fig. 9a). The frame-wise D_{PSNR} values of VASLA-ME in all frames of two videos are better than all the compared block-matching algorithms. On the other hand, frame-wise D_{PSNR} values of PLA-ME, TPLA-ME, and DS are close together.

D_{PSNR} values of evolutionary algorithms-based methods (PSO-BM, DE-BM, and HS-BM) are close to the proposed algorithms (PLA-ME, TPLA-ME, and VASLA-ME). These methods select locations on the search window that iteratively minimize the error function. We can compare the performance of these methods with the performance of our algorithms in terms of convergence speed and finding the global minimum in the search window. For this comparison, some template blocks with high motion content are selected from Foreman and Stefan video sequences. For these template blocks, we perform the proposed algorithms (PLA-ME, TPLA-ME, and VASLA-ME) along with PSO-BM, DE-BM, and HS-BM to find the best match in the search window. Figure 10 represents the normalized SAD values for the best matches that have been found in different iteration of the algorithms. As it can be observed, the DE-BM and HS-BM converge to a local minimum of search window quickly. The PSO-BM also has a similar behavior as DE-BM and HS-BM. PLA-ME, TPLA-ME, and VASLA-ME have relatively slow and incremental behavior. In the Foreman video sequence, the three proposed algorithms converge to the global minimum in the search window. In the Stefan video sequence, only VASLA-ME converges to the global minimum while TPLA-ME and PLA-ME are very close to VASLA-ME in term of SAD value. This comparison indicates that in a video with high motion content the suggested algorithms have a better performance in comparison with evolutionary algorithms-based methods.

4.2 Search efficiency

We consider search efficiency criterion as a measure of computational complexity [19]. The search efficiency criterion is

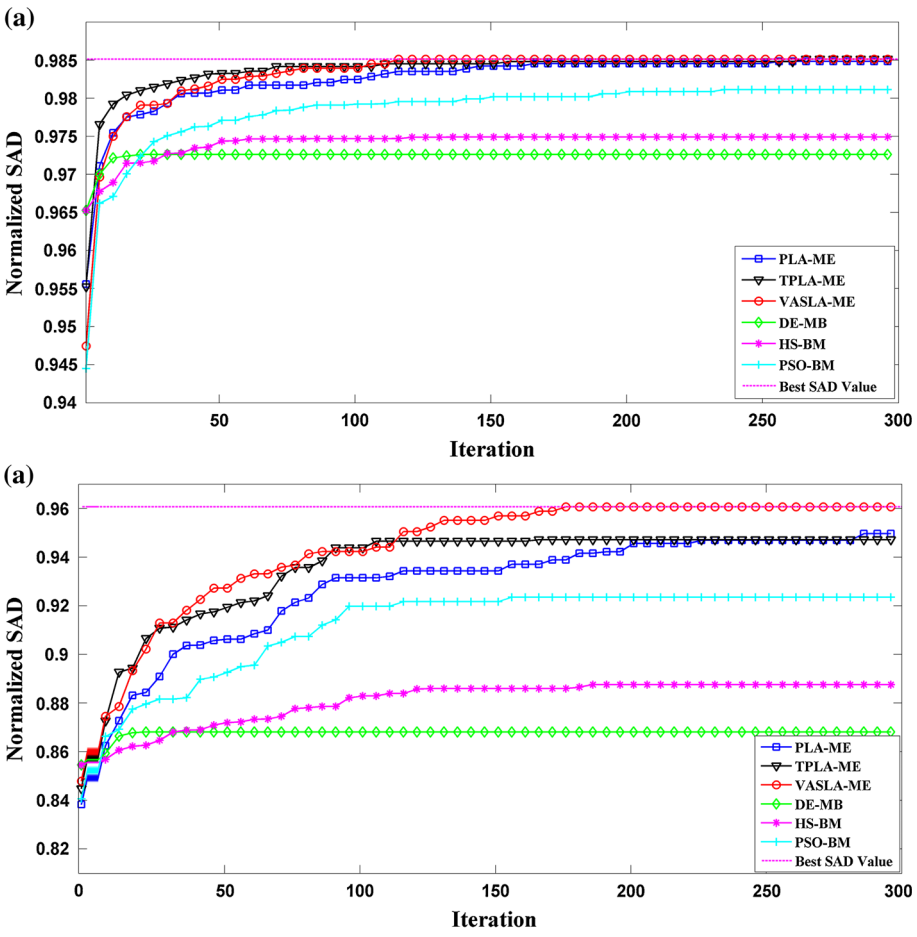

Fig. 10 Normalized SAD values for the best found matching in different iteration of the algorithms for a high motion block in "Foreman" sequence (**a**) and "Stefan" sequence (**b**). The proposed algorithm converge faster to the best achievable SAD value

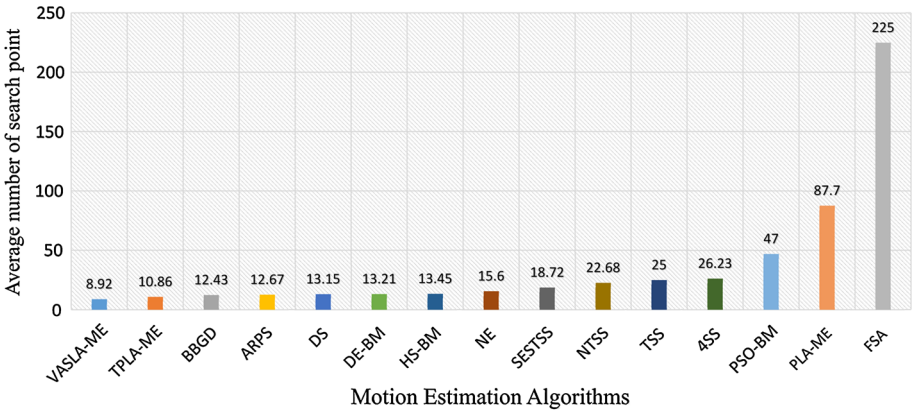

Fig. 11 Mean number of search locations averaged over the all the six considered sequences. VASLA-ME and TPLA-ME have the best performances

the average number of search points (i.e., the average number of SAD computations) for motion vectors estimation. In other words, this metric measures the improvement in complexity of the motion estimation algorithm compared to FSA.

Table 3 shows the search efficiency; FSA, PLA-ME, and PSO-BM hold the highest number of search points per block. The average number of search locations over the six considered sequences is presented in Fig. 11. According to the obtained values, the suggested VASLA-ME and TPLA-ME methods demonstrate a better performance in comparison with the other motion estimation algorithms. The mean number of checked search points by the VASLA-ME and TPLA-ME are 8.92 and 10.86, respectively. According to the last column of Table 3, these two methods have the best ranks

Table 3 Average number of search points per block for all motion estimation methods

Algorithm	Container	Akiyo	Carphone	Foreman	Stefan	Tennis	Total average	Rank
FSA	225	225	225	225	225	225	225	15
TSS	25	25	25	25	25	25	25	11
4SS	19	27.3	25.5	24.8	29.3	31.5	26.23	12
NTSS	17.2	23.5	21.8	22.1	25.4	26.1	22.68	10
SESTSS	19.20	19.20	18.86	18.85	17.45	18.8	18.72	9
ARPS	8.91	8.87	10.99	11.43	18.21	17.65	12.67	4
BBGD	8.1	10.2	11.5	12.5	15.2	17.1	12.43	3
DS	7.5	11.8	12.5	13.4	16.2	17.5	13.15	5
NE	11.7	14.5	13.8	14.2	19.2	20.2	15.6	8
PSO-BM	32.5	48.5	48.5	48.1	52.2	52.2	47	13
DE-BM	9.2	12.5	12.2	12.5	16.1	16.8	13.21	6
HS-BM	8	12.2	12.5	12.2	17.1	16.87	13.45	7
PLA-ME	75.27	49.7	71.54	77.5	131.7	120.5	87.70	14
TPLA-ME	10.85	8.69	9.17	10.09	13.91	12.5	10.86	2
VASPLA-ME	**9.31**	**5.54**	**7.57**	**6.62**	**12.9**	**11.6**	**8.92**	**1**

in search efficiency. The number of search points can be significantly reduced by the proposed approaches (VASLA-ME and TPLA-ME).

The frame-wise average number of search points for PLA-ME, TPLA-ME, and VASLA-ME along with TSS, NTSS, SESTSS, 4SS, DS, ARPS are compared for medium motion Foreman (Fig. 8b) and high motion Stefan (Fig. 9b). In terms of the mean number of search points, only two algorithms VASLA-ME and TPLA-ME perform better than the other compared algorithms.

It should be pointed out that in VASLA-ME, we use the idea for fast calculation of block-sum pyramid [35]. In our implementation, the overhead of construction of the block-sum pyramid for all candidate blocks in the search window is approximately equivalent to eight search points by assuming block size of 16*16, in other words, two search points for each level of the pyramid. In VASLA-ME, on average, 97 candidate blocks are eliminated during the learning process. 83 % of these candidate blocks removed at the top level of the pyramid, 14 % removed in the second level of the pyramid, and eventually for 3 % of these candidate blocks the third the level of pyramid are calculated. Therefore, the overhead of these calculations approximately is equivalent to 2.4 search points.

5 Discussion

As mentioned earlier, the proposed algorithms select a subset of search points to minimize the error function. This selection is performed by learning automata. The third group of algorithms, which we mentioned in Sect. 1, reduce the computational cost of the error functions. In [22], a winner-update strategy is utilized to avoid unnecessary computation of the search positions, which can reduce a large portion of the computation and guarantee finding the globally optimal solution. In their algorithm, only the current winner position with the minimal accumulated error is considered for updating the accumulated error function. This algorithm examines all the candidate blocks in the search window, but skips the unnecessary calculations of the error function for most candidates. In fact, the results of the algorithm are not exactly the same as FSA. When two or more search positions have the same minimum error function, the result depends on the order of scan. VASLA-ME combines the techniques in the first and the second categories (Sect. 1). In VASLA-ME, some of the search points are selected by the LA and partial matching error is computed by applying the hierarchical pyramid and multi-resolutional Minkowski inequality. This combination of two approaches in a single algorithm has increased the ability of the algorithm for motion estimation.

6 Conclusion and future works

In this study, we proposed three novel block-matching algorithms based on learning automata (PLA-ME, TPLA-ME, and VASLA-ME) for motion estimation. We compared performances of these algorithms to some of the other well-known block-matching algorithms. Experimental results demonstrate superior prediction quality for all the proposed algorithms. Moreover, two of the proposed algorithms are using less search points to find the best match,

thus, they are more efficient than the other block-matching algorithms. VASLA-ME and TPLA-ME on average only check 8.92 and 10.86 search point, respectively. In this paper, we applied LA to a basic machine vision problem and we achieved promising results. For future works, we suggest extending the application of LA to other machine vision problems such as image inpainting and image segmentation.

References

1. Stuhlmüller, K., Färber, N., Link, M., Girod, B.: Analysis of video transmission over lossy channels. Sel. Areas Commun. IEEE J. **18**(6), 1012–1032 (2000)
2. Courtney, J.D.: Automatic video indexing via object motion analysis. Pattern Recognit. **30**(4), 607–625 (1997)
3. Zhang, H.J., Kankanhalli, A., Smoliar, S.W.: Automatic partitioning of full-motion video. Multimed. Syst. **1**(1), 10–28 (1993)
4. Koga, T.: Motion-compensated interframe coding for video conferencing. Proc. NTC **81**, C9–6 (1981)
5. Kratz, S., Ballagas, R.: Gesture recognition using motion estimation on mobile phones. In: Proceedings of 3rd International Workshop on Pervasive Mobile Interaction Devices (PERMID'07) (2007)
6. Chua, C.-S., Guan, H., Ho, Y.-K.: Model-based 3d hand posture estimation from a single 2d image. Image Vis. Comput. **20**(3), 191–202 (2002)
7. Sarrut, D., Delhay, B., Villard, P.-F., Boldea, V., Beuve, M., Clarysse, P.: A comparison framework for breathing motion estimation methods from 4-d imaging. Med. Imag. IEEE Trans. **26**(12), 1636–1648 (2007)
8. Wang, Yao: Motion Estimation for Video Coding. Polytechnic University, Brooklyn (2003)
9. Yaakob, R., Aryanfar, A., Halin, A.A., Sulaiman, N.: A comparison of different block matching algorithms for motion estimation. Proc. Technol. **11**, 199–205 (2013)
10. Barjatya, A.: Block matching algorithms for motion estimation. IEEE Trans. Evol. Comput. **8**(3), 225–239 (2004)
11. Jong, H.-M., Chen, L.-G., Chiueh, T.-D.: Accuracy improvement and cost reduction of 3-step search block matching algorithm for video coding. Circuits Syst. Video Technol. IEEE Trans. **4**(1), 88–90 (1994)
12. Li, R., Zeng, B., Liou, Ming L.: A new three-step search algorithm for block motion estimation. Circuits Syst. Video Technol. IEEE Trans. **4**(4), 438–442 (1994)
13. Lu, J., Liou, M.L.: A simple and efficient search algorithm for block-matching motion estimation. Circuits Syst. Video Technol. IEEE Trans. **7**(2), 429–433 (1997)
14. Po, L.-M., Ma, W.-C.: A novel four-step search algorithm for fast block motion estimation. Circuits Syst. Video Technol. IEEE Trans. **6**(3), 313–317 (1996)
15. Zhu, S., Ma, Kai-Kuang: A new diamond search algorithm for fast block-matching motion estimation. Image Process. IEEE Trans. **9**(2), 287–290 (2000)
16. Nie, Y., Ma, Kai-Kuang: Adaptive rood pattern search for fast block-matching motion estimation. Image Process. IEEE Trans. **11**(12), 1442–1449 (2002)
17. Liu, L.-K., Feig, Ephraim: A block-based gradient descent search algorithm for block motion estimation in video coding. Circuits Syst. Video Technol. IEEE Trans. **6**(4), 419–422 (1996)
18. Saha, A., Mukherjee, J., Sural, S.: A neighborhood elimination approach for block matching in motion estimation. Signal Process: Image Commun. **26**(8), 438–454 (2011)
19. Cuevas, E., Zaldívar, D., Pérez-Cisneros, M., Oliva, D.: Block-matching algorithm based on differential evolution for motion estimation. Eng. Appl. Artif. Intell. **26**(1), 488–498 (2013)
20. Saha, A., Mukherjee, J., Sural, Shamik: New pixel-decimation patterns for block matching in motion estimation. Signal Process. Image Commun. **23**(10), 725–738 (2008)
21. Li, W., Salari, Ezzatollah: Successive elimination algorithm for motion estimation. Image Process. IEEE Trans. **4**(1), 105–107 (1995)
22. Chen, Y.-S., Hung, Y.-P., Fuh, C.-S.: Fast block matching algorithm based on the winner-update strategy. Image Process. IEEE Trans. **10**(8), 1212–1222 (2001)
23. Zahiri, S.-H.: Learning automata based classifier. Pattern Recognit. Lett. **29**(1), 40–48 (2008)
24. Sang, Q., Lin, Z., Acton, ST.: Learning automata for image segmentation. Pattern Recognit. Lett. **74**, 46–52 (2016)
25. Vahidipour, S.M., Meybodi, M.R., Esnaashari, M.: Learning automata-based adaptive petri net and its application to priority assignment in queuing systems with unknown parameters. IEEE Trans. Syst. Man Cybern. Syst. **45**(10), 1373–1384 (2015)
26. Rezvanian, A., Meybodi, MR.: A new learning automata-based sampling algorithm for social networks. Int. J. Commun. Syst. Wiley (2015). doi:10.1002/dac.3091
27. Narendra, KS., Thathachar, MAL.: Learning Automata: An Introduction. Dover Publications, Inc. Mineola, New York (2012)
28. Narendra, KS., Thathachar, MLAA.: Learning automata-a survey. Syst. Man Cybern. IEEE Trans. (4):323–334 (1974)
29. Thathachar, M., Sastry, P.S.: Varieties of learning automata: an overview. Syst. Man Cybern. Part B: Cybern. IEEE Trans. **32**(6), 711–722 (2002)
30. Phansalkar, V.V., Thathachar, M.A.L.: Local and global optimization algorithms for generalized learning automata. Neural Comput. **7**(5), 950–973 (1995)
31. Thathachar, M.A.L., Sastry, P.S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Springer, Berlin (2011)
32. Thathachar, MAL., Sastry, PS.: Estimator algorithms for learning automata. In: Proceedings of the Platinum Jubilee Conference on system Signal Processing, Department of Electrical Engineering, Indian Institute of Science, Bangalore, India, December 1986
33. Oommen, B.J., Lanctôt, J.K.: Discretized pursuit learning automata. Syst. Man Cybern. IEEE Trans. **20**(4), 931–938 (1990)
34. Thathachar, M.L., Harita, B.R.: Learning automata with changing number of actions. Syst. Man Cybern. IEEE Trans. **17**(6), 1095–1100 (1987)
35. Lee, C.-H., Chen, L.-H.: A fast motion estimation algorithm based on the block sum pyramid. Image Process. IEEE Trans. **6**(11), 1587–1591 (1997)
36. Zhang, J.Q., Wang, C., Zhou, M.C.: Fast and epsilon-optimal discretized pursuit learning automata. Cybern. IEEE Trans. **45**(10), 2089–2099 (2015)
37. Yuan, X., Shen, X.: Block matching algorithm based on particle swarm optimization for motion estimation. In: Embedded Software and Systems, 2008. ICESS'08. International Conference on, pp. 191–195. IEEE (2008)

Bahman Damerchilu received the BSc degree in Information Technology from the Institute for Advanced Studies in Basic Sciences, Zanjan, Iran, in 2012, and the MSc degree in Artificial Intelligence from the Amirkabir University of Technology, Tehran, Iran, in 2015. His current research interests are learning automata, machine learning, image processing, machine vision, and artificial intelligence in medicine.

Mohammad Sadegh Norouzzadeh received his BSc degree in Software Engineering from Kharazmi University, Iran, in 2007 and the MSc degree in Artificial Intelligence from Isfahan University of Technology, in 2009. Currently, he is a PhD student of Computer Science at the University of Wyoming, Laramie, Wyoming, USA. His research interests include machine learning, evolutionary computations, and image processing.

Mohammad Reza Meybodi received the BSc and MSc degrees in Economics from Shahid Beheshti University, Tehran, Iran, in 1973 and 1977, respectively, the MSc and PhD degrees in Computer Science from Oklahoma University, Norman, OK, USA, in 1980 and 1983, respectively. He was an Assistant Professor at Western Michigan University, Kalamazoo, MI, USA, from 1983 to 1985, and an Associate Professor at Ohio University, Athens, OH, USA, from 1985 to 1991. He is currently a Full Professor at the Computer Engineering Department, Amirkabir University of Technology, Tehran. His research interests include wireless networks, faulttolerant systems, learning systems, parallel algorithms, soft computing, and software development.