Parallel Appearance-Adaptive Models for Real-Time Object Tracking Using Particle Swarm Optimization

Boguslaw Rymut and Bogdan Kwolek

Rzeszów University of Technology
W. Pola 2, 35-959 Rzeszów, Poland
{brymut,bkwolek}@prz.edu.pl

Abstract. This paper demonstrates how appearance adaptive models can be employed for real-time object tracking using particle swarm optimization. The parallelization of the code is done using OpenMP directives and SSE instructions. We show the performance of the algorithm that was evaluated on multi-core CPUs. Experimental results demonstrate the performance of the algorithm in comparison to our GPU based implementation of the object tracker using appearance-adaptive models. The algorithm has been tested on real image sequences.

Keywords: Swarm intelligence, particle swarm optimization.

1 Introduction

One of the rising stars of Collective Intelligence is Particle Swarm Optimization (PSO), developed in 1995 by Kennedy and Eberhart [10]. PSO is a derivative-free optimum search algorithm based on the collective intelligence of a group of simple agents, which interact with each other and with their environment. Such local interactions result in the global behavior of the whole population. The individual entities are simple, knowing no more than their own current locations and fitness scores, their personal best locations, and the swarm's best location. In Particle swarm optimization, each potential solution to the problem is called particle and the word "swarm" comes from the irregular movements of the particles in the problem space. PSO was inspired by the social behavior of bird flocking and fish schooling and it has its roots in artificial life and social psychology, as well as in engineering and computer science.

PSO can be used to solve a wide spectrum of different optimization problems, including tasks that can be solved using Genetic Algorithms. Some example applications include function minimization and neural network training. In [13], a parallel PSO algorithm was proposed. The algorithm is synchronous and has good performance on problems where the fitness evaluations require the same amount of time. In [1], a PSO algorithm was successfully applied to object tracking. In [12] it has been shown that a GPU implementation of a PSO based object tracker can exhibit a more than 40-fold speed-up over a CPU implementation.

Vision based object tracking is a mandatory step in many applications, such as surveillance, traffic monitoring, sport event analysis, mixed virtual reality and even in observational problems in the natural science. It is one of the major steps toward understanding video content. The goal of visual object tracking is to repeatedly localize an object of interest in successive frames. Most object trackers search for the target locally in new frames using a similarity measure between a reference object model and candidate targets. Thus, the task of object tracking can be considered as a numerical optimization problem, where a local optimization is used to track the local mode of the similarity measure in a parameter space of translation, rotation and scale. In [14], it was shown that in tasks consisting in tracking the face or the human a PSO tracker outperforms a particle filter based tracker in terms of accuracy. Particle filters [8], which are sequential Monte Carlo methods based on point mass representations of probability densities, can be employed to any state-space model and generalize the conventional Kalman filtering methods. Currently, they are widely used to track targets in image sequences. Their weakness is that they require a large number of particles for accurate estimation of state variables lying in a high dimensional space. In contrary, PSO has better capabilities to explore the search space as well as facility to balance the trade-off between exploration and exploitation.

Image sequences acquired in real scenarios pose specific challenges to tracking algorithms, due to, for example, low or variable lighting conditions, scale change and changes of the appearance. On-line appearance models [9] are one of the most successful approaches to object tracking. An on-line variant of the Expectation Maximization (EM) [5] algorithm is typically used to learn the parameters of the appearance models. It identifies stable structures and naturally combines such structures with transient image information in the $\mathcal{WSL}$ framework [9]. Adaptive appearance models were successfully applied in various applications, including challenging tasks like model based articulated object tracking [2][11]. One of the drawbacks of the object trackers, which are built on adaptive appearance models is considerable computation time. This motivated us to develop the GPU implementation of a tracker built on the appearance-adaptive models and the particle swarm optimization [12].

In this work we show the processing times of our GPU object tracker, which was extended about the affine transformations. We demonstrate the computation times as well as speeds-up that have been obtained on GPU as well as on multi-core CPUs. The parallel computations on multi-core CPUs were achieved using Streaming SIMD Extensions (SSE) and Open Multi-Processing (OpenMP). Currently, OpenMP is widely utilized standard for parallelizing programs in a shared memory environment [3]. Experimental results show that our parallel algorithm exhibits about 4.5 fold speed-up over standard C/C++ implementation. Using 256 and 512 particles in PSO the tracking at a 4-core CPU can be done with 20 and about 10 frames per second, respectively. Using a 2-core CPU of a low-cost notebook we tracked objects with 6 fps. The algorithm has been tested on real image sequences.

2 Appearance-Adaptive Models for Object Tracking

Let $I_t(x)$ denote the brightness value at the location $x = [x_1, x_2]^T$ in an image $\mathcal{I}$ that was acquired at time t. Let $\mathcal{R}$ be a set of J image locations $\{x(j) \,|\, j = 1, 2, ..., J\}$ defining a template. $Y_t(\mathcal{R}) = \{I_t(x(j)) \,|\, j = 1, 2, ..., J\}$ is a vector of the brightness values at locations $x(j)$ in the template. We assume that the transformations of the template can be modeled by a parametric motion model $g(x; \omega_t)$, where x denotes an image location and ω_t is a motion parameter vector.

The image variations of planar objects that undergo orthographic projection can be described by a six-parameter affine motion models [6]:

$$g(x; \omega) = \begin{bmatrix} a & d \\ c & e \end{bmatrix} x + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = Ax + u, \tag{1}$$

where the motion parameter vector $\omega = (a, c, d, e, u_1, u_2)^T$. The affine transformation allows different stretching along rows and columns of an image and shearing. Applying an affine transformation on image patches is called warping. We assume that the warped current image is a representation of the reference object template. The goal of the tracking is to estimate the warping parameters $\hat{\omega}_t$ in each frame.

Our intensity-based appearance model consists of three components, namely, The $\mathcal{W}$-component accounting for the two-frame variation, the $\mathcal{S}$-component depicting the stable structure within all previous observations and $\mathcal{F}$ component, which takes the place of the original lost component $\mathcal{L}$ and represents a fixed template. The model $A_t = \{\mathcal{W}_t, \mathcal{S}_t, \mathcal{F}_t\}$ represents the appearances existing in all observations up to time $t - 1$. It is a mixture of Gaussians [9] with centers $\{\mu_{i,t} \,|\, i = w, s, f\}$, their corresponding variances $\{\sigma^2_{i,t} \,|\, i = w, s, f\}$ and mixing probabilities $\{m_{i,t} \,|\, i = w, s, f\}$.

The fitness score has been evaluated according to the following equation:

$$f(\omega_t) = \prod_{j=1}^{J} \sum_{i=w,s,f} \frac{m_{i,t}(j)}{\sqrt{2\pi\sigma^2_{i,t}(j)}} \exp\left[-\frac{1}{2}\left(\frac{\check{Y}_t(j) - \mu_{i,t}(j)}{\sigma_{i,t}(j)}\right)^2\right] \tag{2}$$

where $\check{Y}_t(\mathcal{R}) = Y_t(g(\mathcal{R}; \omega_t))$. In the fitness function we utilize a recursively updated appearance model, which depicts stable structures seen so far, initial object appearance as well as two-frame variations.

The update of the current appearance model A_t to A_{t+1} is done using the EM algorithm. For a template $\hat{Y}_t(\mathcal{R}) = Y_t(g(\mathcal{R}; \hat{\omega}_t))$, which has been obtained through applying the estimated $\hat{\omega}_t$, we evaluate the posterior contribution probabilities as follows:

$$o_{i,t}(j) = \frac{m_{i,t}(j)}{\sqrt{2\pi\sigma^2_{i,t}(j)}} \exp\left[-\frac{1}{2}\left(\frac{\hat{Y}_t(j) - \mu_{i,t}(j)}{\sigma_{i,t}(j)}\right)^2\right] \tag{3}$$

where $i = w, s, f$ and $j = 1, 2, ..., J$. The posterior contribution probabilities (with $\sum_i o_{i,t}(j) = 1$) are utilized in updating the mixing probabilities in the

following manner:

$$m_{i,t+1}(j) = \gamma o_{i,t}(j) + (1-\gamma) m_{i,t}(j) \quad | \, i = w, s, f \tag{4}$$

where γ is accommodation factor. Then, the first and the second-moment images are determined as follows:

$$M_{1,t+1}(j) = (1-\gamma) M_{1,t}(j) + \gamma o_{s,t}(j) \hat{Y}_t(j) \tag{5a}$$

$$M_{2,t+1}(j) = (1-\gamma) M_{2,t}(j) + \gamma o_{s,t}(j) \hat{Y}_t^2(j) \tag{5b}$$

In the last step the mixture centers and the variances are calculated as follows:

$$\mu_{s,t+1}(j) = \frac{M_{1,t+1}(j)}{m_{s,t+1}(j)}, \;\; \sigma_{s,t+1}(j) = \sqrt{\frac{M_{2,t+1}(j)}{m_{s,t+1}(j)} - \mu_{s,t+1}^2(j)} \tag{6}$$

$$\mu_{w,t+1}(j) = \hat{Y}_t(j), \quad \sigma_{w,t+1}(j) = \sigma_{w,1}(j) \tag{7}$$

$$\mu_{f,t+1}(j) = \mu_{t,1}(j), \quad \sigma_{f,t+1}(j) = \sigma_{f,1}(j) \tag{8}$$

In order to initialize the model A_1 the initial moment images are set using the following formulas: $M_{1,1} = m_{s,1} Y_{t0}(\mathcal{R})$ and $M_{2,1} = m_{s,1}(\sigma_{s,1}^2 + Y_{t0}^2(\mathcal{R}))$.

3 Object Tracking Using PSO

PSO is a population based algorithm introduced in [10] that utilizes a set of particles representing potential solutions of the optimization task. Despite the simplicity of the individual particles, the swarm as a whole has a remarkable level of coherence and coordination. Each solution is represented as a series of coordinates in n-dimensional space. A number of particles are initialized randomly within the search space. Every particle flies in the solution space with a velocity adjusted dynamically according to its own experience and the experience of the whole swarm. Each particle has a very simple memory of its personal best solution so far, called *pbest*. The global best solution for each iteration is also determined and is termed *gbest*. On each iteration, every particle is moved a certain distance from its current location, influenced a random amount by the *pbest* and *gbest* values. The particles are evaluated according to a user defined fitness function $f()$. The velocity of each particle i is updated in accordance with the following equation:

$$v_i^{(j)} \leftarrow w v_i^{(j)} + c_1 r_1^{(j)} (pbest_i^{(j)} - \omega_i^{(j)}) + c_2 r_2^{(j)} (gbest_i - \omega_i^{(j)}) \tag{9}$$

where $v_i^{(j)}$ is the velocity in the $j-$th dimension of the $i-$th particle, c_1, c_2 denote the acceleration coefficients, $r_1^{(j)}$ and $r_2^{(j)}$ are uniquely generated random numbers in the interval $[0.0,\ 1.0]$, and w stands for an inertia weight. The inertia

weight allows the balance of the exploration and exploitation abilities of the swarm as well as eliminates the need for velocity clamping.

The first part in (9) takes into account the previous velocity, which provides the necessary momentum for particles to fly across the search space. The second part is known as the cognitive component and represents the personal thinking of each particle. This component encourages the particles to fly toward their own best position *pbest* found so far. The third part is known as the social component and represents the collaborative effect of the particles in finding the global optimum. This component pulls the particles toward the best position(s) found so far by their neighbors. The inertia part keeps particles to explore new areas while the cognitive and social parts try to keep them exploiting around the visited points.

The new position of a particle is calculated in the following manner:

$$x_i^{(j)} \leftarrow x_i^{(j)} + v_i^{(j)} \tag{10}$$

The local best position of each particle is updated as follows:

$$pbest_i \leftarrow \begin{cases} \omega_i, & \text{if } f(\omega_i) > f(pbest_i) \\ pbest_i, & \text{otherwise} \end{cases} \tag{11}$$

and the global best position *gbest* is defined as:

$$gbest \leftarrow \arg\max_{pbest_i} \{f(pbest_i)\} \tag{12}$$

The value of velocity v_i should be restricted to the range $[-v_{max}, v_{max}]$ to prevent particles from moving out of the search range. In [10], Eberhart and Kennedy suggested that the PSO should perform better if v_{max} in each dimension is set equal to the dynamic range of that dimension. In some optimization problems the local best version of PSO, where particles are influenced by the best position within their neighborhood, as well as their own past experience can give better results. While such a configuration of the PSO is generally slower in convergence than algorithm with *gbest*, it typically results in much better solutions and explores a larger part of the problem space.

At the beginning of the optimization the PSO initializes randomly locations as well as the velocities of the particles. Then the algorithm selects *pbest* and *gbest* values. Afterwards, equations (9)-(12) are called until maximum iterations or minimum error criteria is attained. After that, given $\hat{\omega}_t = gbest$ we calculate $\hat{Y}_t$, and then update of the object model using formulas (3)-(8).

In contrast to traditional optimization problems with stationary optima, tracking objects in image sequences requires the algorithm to find the optimum not once, but in every successive image. There are various approaches to dealing with moving objects, such as decaying the score of the best location after every. In consequence, such an operation results in forcing the swarm to continually search for a better location. In particular, it prevents the swarm from completely converging to a single point, allowing the swarm agents to be appropriately spaced in order to quickly reacquire a target in the next image.

In the simplest solution the tracking can be realized as deterministic searching of window location whose content best matches a reference window content. PSO allows us to avoid such time consuming exhaustive searching for the best match. It provides an optimal or sub-optimal match without the complete knowledge of the searching space. In PSO based tracking, at the beginning of each frame in the initialization stage, an initial position is assigned to each particle

$$\omega_{i,t} \leftarrow \mathcal{N}(gbest, \Sigma) \tag{13}$$

given the location *gbest* that has been estimated in the previous frame $t-1$. In the evaluation phase the fitness value of each particle is determined on the basis of (2). Every particle has an associated affine transformation, which is used to warp a video frame.

4 Experimental Results

The experiments were conducted on a desktop PC with 4 GB RAM, Intel Core i5, 2.8 GHz processor with NVIDIA GeForce 9800 GT graphics card. The graphics card has 14 stream multiprocessors with 1.5 GHz, each with 8 cores. It is equipped with 1024 MB RAM, 64 KB constant memory and 16 KB common memory for each multiprocessor. We implemented the algorithm in CUDA and compared the runtimes with its counterpart that was implemented in C/C++ and executed on the CPU. The CPU code was compiled with Intel C++ Compiler for Windows. Table 1 shows the running times of the tracking algorithm both on CPU and GPU as well as the speed-up. The communication delays for copying images from CPU to GPU and vice versa have not been taken into account. The most time-consuming operation of the algorithm is calculation of the fitness function (2). This operation amounts to 0.9 of the whole processing time.

Table 1. Tracking time [ms] and speed-up of GPU (NVIDIA GeForce 9800 GT) over CPU (Intel Core i5, 2.8 GHz) at a desktop PC.

# particles	256	512	1024	2048	4096
CPU [ms]	250	405	949	1878	3737
GPU [ms]	61	69	72	90	165
CPU/GPU	4.1	5.7	13.2	20.9	22.7

OpenMP is a library (application program interface or API) that supports parallel programming on shared memory parallel computers. It consists of a set of directives (pragmas) and library routines that can be inserted into Fortran or C/C++ codes to enable use of more than one thread. It handles scheduling over available cores in a static fashion. OpenMP provides a fork-and-join execution model in which a program begins execution as a thread. The thread executes

sequentially until a parallelization directive for a structured block of code is found. If this takes place, such a thread creates a set of threads and becomes the master thread of the new group of threads. Each thread executes the same code redundantly until the end of the parallel section and the threads communicate by sharing variables. The exit point of a structured block is an implicit synchronization point for the master thread and the threads created for the block. After the synchronization the master thread continues with the computation and the other threads end. The advantage of OpenMP is that an existing code can be effortlessly parallelized by placing OpenMP directive instructions.

Table 2 contains tracking times that have been obtained using OpenMP. As we can see the efficiency of parallel computations is very high. The speed-up over the program executed without OpenMP support is about 3.3. The speed-up was achieved owing to data parallelism [7], which is also known as loop-level parallelism. It focuses on effective distributing the data across different parallel computing nodes. In the appearance-adaptive models the pixels are assumed to be independent and therefore they can be processed in parallel.

Table 2. Tracking time [ms] and speed-up of the GPU over the CPU (Intel Core i5, 2.8 GHz, OpenMP).

#particles	256	512	1024	2048	4096
CPU [ms]	73	145	284	573	1142
CPU/GPU	1.2	2.1	4.0	6.4	6.9

Table 3 shows computation times that were achieved using OpenMP and SSE instructions and registers. The functions like `exp`, `log`, `pow` were implemented using algorithms presented in [4] and SSE, SSE2 SSE3 and SSSE 3 (Supplemental Streaming SIMD Extension 3) instructions. All SSE instructions operate on 128-bit XMM registers. By the use of such registers the fitness score given by (2) can be computed very quickly. In our implementation we evaluate simultaneously all three Gaussians $\mathcal{N}(\check{Y}, \mu_i, \sigma_i)$, which are then simultaneously multiplied by the mixing probabilities. Such values are finally used in updating the fitness score. This process can be depicted in the following pseudo-code:

$$
\begin{aligned}
f_w &= m_w \times \mathcal{N}(\check{Y}, \mu_w, \sigma_w) \\
f_s &= m_s \times \mathcal{N}(\check{Y}, \mu_s, \sigma_s) \\
f_f &= m_f \times \mathcal{N}(\check{Y}, \mu_f, \sigma_f) \\
f &\leftarrow f + \texttt{log}(f_w + f_s + f_f)
\end{aligned}
$$

For each mixture j the mixing probabilities m_w, m_s, m_f are stored in 96 bits of single 128-bit XMM register. The corresponding mixture centers and the variances are stored in two XMM registers.

As we can see in Tab. 3 the speed-up is considerable. For 512 particles the algorithm is faster than the GPU algorithm. It is worth noting that for such a number of particles not all GPU resources were fully exploited.

Table 3. Tracking time [ms] and speed-up of the GPU over the CPU (Intel Core i5, 2.8 GHz, SSE and OpenMP).

#particles	256	512	1024	2048	4096
CPU [ms]	51	101	200	398	803
CPU/GPU	0.8	1.5	2.8	4.4	4.9

In Tab. 4 are presented the results that were obtained on a typical notebook with Intel Core 2 Duo 2.2 CPU and NVIDIA GeForce 9600M GS graphics card. As we can notice, the ratio of computation times for 4096 and 256 particles at the notebook is far larger comparison of the ratio on the desktop computer. The graphics card of the desktop has 14 multiprocessors, whereas the graphics card of the notebook has 4 multiprocessors and therefore the discussed ratio is smaller for the desktop computer.

Table 4. Tracking time [ms] and speed-up of GPU (NVIDIA GeForce 9600M GS) over CPU (Intel Core 2 Duo, 2.2 GHz, OpenMP and SSE) at a notebook.

#particles	256	512	1024	2048	4096
CPU [ms]	164	339	667	1327	2651
GPU [ms]	65	84	113	248	416
CPU/GPU	2.5	4.0	5.9	5.4	6.4

The experimental results presented above indicate that OpenMP together with SSE instructions can lead to faster tracking algorithm on multi-core CPUs. More important, thanks to OpenMP support we achieved tracking in real-time using 265 particles, i.e. tracking with 13 fps, see Tab. 2. With OpenMP and SEE support we performed tracking using 512 particles at about 10 Hz, see also results in Tab. 3.

Figure 1 depicts some tracking results that were achieved using an active camera. The aim of the algorithm was to keep the face undergoing tracking at specific location in the image. In the tracking experiments the face moved in front of the wooden furniture, which has very similar color to skin color. It is worth to note that in such a scenario very popular skin-color based trackers are unable to track the target. The precision of tracking in the depicted image sequence was about 0.5 pix using PSO consisting of 256 particles. The precision was evaluated using the corners of the template as ground-truth, which has been determined manually. The size of the reference template is 32×32 pixels. The algorithm operates on images of size 640×480 pixels.

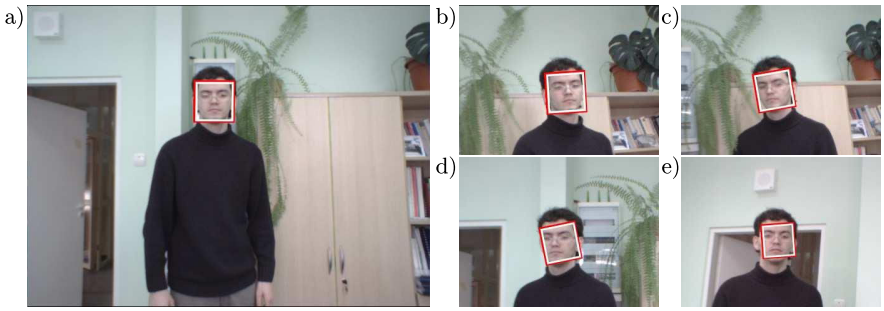

Fig. 1. Real-time tracking of the face using an active camera. Frame #0 a), #50 b), #100 c), #150 d) and #200 e).

We conducted also experiments consisting in person following with a mobile robot. The camera was mounted on a mobile robot Pioneer 2DX. Sample experimental results are depicted in Fig. 2.

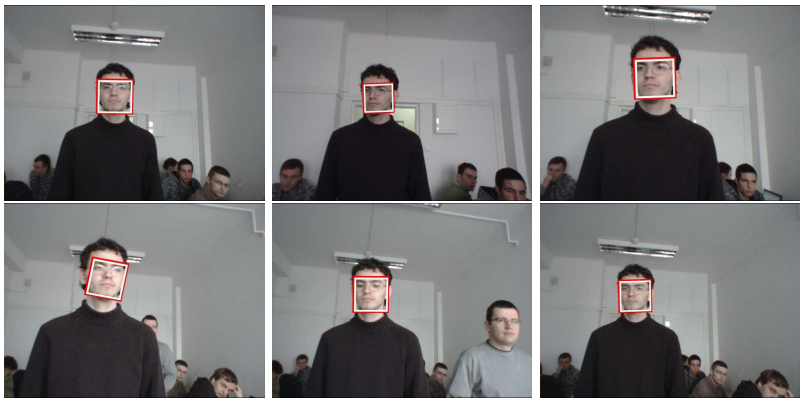

Fig. 2. Person following using a vision-guided mobile robot. Frames #55, 120, 165, 250, 275, 300 (left-to-right, top-to-bottom).

5 Conclusions

In this paper, we have shown how the particle swarm optimization tracker built on appearance-adaptive models can be accelerated significantly using OpenMP and SSE instructions. The results showed that our algorithm running on a 4-core CPU is about 4.5 times faster than an algorithm based on pure C/C++ code. As a result the tracking algorithm runs at frame-rates exceeding 20 frames per

second. Future work will concentrate on particle swarm optimization based multiple object tracking with the use of the appearance-adaptive models. We intend to apply multiple swarms to make collective decisions for objects undergoing temporal occlusions.

Acknowledgement. This work has been partially supported by the National Science Centre (NCN) within the project N N516 483240.

References

1. Anton-Canalis, L., Hernandez-Tejera, M., Sanchez-Nielsen, E.: Particle swarms as video sequence inhabitants for object tracking in computer vision. In: Proc. of the Sixth Int. Conf. on Intelligent Systems Design and Applications Vol. 2. pp. 604–609. ISDA'06, IEEE Computer Society, Washington, DC, USA (2006)
2. Balan, A.O., Black, M.J.: An adaptive appearance model approach for model-based articulated object tracking. In: Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, Vol. 1. pp. 758–765. IEEE Computer Society, Washington, DC, USA (2006)
3. Chapman, B., Jost, G., van der Pas, R., Kuck, D.: Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press (2007)
4. Cody, W.J.: Software Manual for the Elementary Functions (Prentice-Hall series in computational mathematics). Prentice-Hall, Inc., Upper Saddle River, USA (1980)
5. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. of the Royal Statistical Society. Series B 39(1), 1–38 (1977)
6. Hager, G.D., Belhumeur, P.N.: Efficient region tracking with parametric models of geometry and illumination. IEEE Trans. on PAMI 20(10), 1025–1039 (1998)
7. Hillis, W.D., Steele, Jr., G.L.: Data parallel algorithms. Commun. ACM 29, 1170–1183 (December 1986)
8. Isard, M., Blake, A.: Condensation - conditional density propagation for visual tracking. Int. J. of Computer Vision 29, 5–28 (2006)
9. Jepson, A.D., Fleet, D.J., El-Maraghi, T.: Robust on-line appearance models for visual tracking. IEEE Trans. on PAMI 25(10), 1296–1311 (2003)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of IEEE Int. Conf. on Neural Networks. pp. 1942–1948. IEEE Press, Piscataway, NJ (1995)
11. Kwolek, B.: 3D model-based tracking of the human body in monocular gray-level images. In: Proc. of Int. Conf. on Computer Vision/Computer Graphics Collaboration Techniques. pp. 494–505. MIRAGE'07, Springer-Verlag (2007)
12. Rymut, B., Kwolek, B.: GPU-supported object tracking using adaptive appearance models and particle swarm optimization. In: Proc. of Int. Conf. on Computer Vision and Graphics, LNCS. pp. 227–234. Springer-Verlag, Vol. 6375 (2010)
13. Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., George, A.D.: Parallel global optimization with the particle swarm algorithm. Int. J. for Numerical Methods in Engineering 61(13), 2296–2315 (2004)
14. Zhang, X., Hu, W., Maybank, S., Li, X., Zhu, M.: Sequential particle swarm optimization for visual tracking. In: IEEE Int. Conf. on CVPR. pp. 1–8 (2008)