Human-centric Computing and Information Sciences

RESEARCH Open Access

Parallel implementation of color-based particle filter for object tracking in embedded systems

Mai Thanh Nhat Truong and Sanghoon Kim*

*Correspondence: kimsh@hknu.ac.kr
Department of Electrical, Electronic, and Control Engineering, Hankyong National University, Room 205, Administrative Building, 327 Jungang-ro, Anseong-si, Gyeonggi-do, South Korea

Abstract

Recently, embedded systems have become popular because of the rising demand for portable, low-power devices. A common task for these devices is object tracking, which is an essential part of various applications. Until now, object tracking in video sequences remains a challenging problem because of the visual properties of objects and their surrounding environments. Among the common approaches, particle filter has been proven effective in dealing with difficulties in object tracking. In this research, we develop a particle filter based object tracking method using color distributions of video frames as features, and deploy it in an embedded system. Because particle filter is a high-complexity algorithm, we utilize computing power of embedded systems by implementing a parallel version of the algorithm. The experimental results show that parallelization can enhance the performance of particle filter when deployed in embedded systems.

Keywords: Object tracking, Particle filter, Parallel computing, Embedded systems

Background

Object tracking has important roles in many vision-based applications such as traffic monitoring [1], surveillance systems [2], and recently, augmented reality [3]. Generally, the goal of object tracking algorithms is to locate the moving objects of interest in the video data retrieved from image acquisition devices and to produce a record of the trajectory of the objects. In practice, object tracking can be more difficult because of object shapes, light conditions, occlusions, sudden change in object motions, camera motions, etc. Owing to various difficulties that cannot be solved simultaneously, object tracking methods are usually designed to track objects with specific properties in certain environments [4]. For instance, one object tracker produces good results in various environments under different lighting conditions, but produces low accuracy results when the target shape or silhouette is changed because of camera angles. One tracking method can predict target movement accurately, but it may fail when tracking bouncing objects as a result of sudden changes in movement direction. Therefore, object tracking is still a high-complexity and time-consuming task. The complexity is increased when the tracking task is performed in environments with complex surroundings, or the requirement is to track objects with various appearances.

Approaches for object tracking can be categorized into three groups: point tracking, kernel tracking, and silhouette tracking [5]. In point tracking, the target objects are represented by points that contain information of the object properties. Objects are tracked using the relation of the points, and their locations and movements are calculated based on the previous state of these points. However, an external mechanism is required to locate the objects in every frame. Kernel tracking relies on the object shape and appearance, which are called kernels. For instance, the kernel can be a rectangular region or an elliptical shape with an associated histogram. Objects are tracked by calculating the motion of the kernel in consecutive frames. This motion is usually defined as a parametric transformation such as translation, rotation, and affine. In silhouette tracking, the target objects are tracked by their estimated region in each frame. Silhouette tracking approaches use the features extracted from the object region, which are also called models. After having obtained the models, object silhouettes are tracked by using shape matching or contour evolution. Essentially, both these methods can be considered as object segmentation applied in the temporal domain using the object state from the previous frames.

Object tracking is also an important application for embedded systems. Recently, embedded systems and SoC (systems-on-chip) platforms have become popular because of the rising demand for portable, low-power devices. One of the most popular manufacturers of this type of platform is ARM Holdings, the developer of the ARM (Acorn RISC Machine) platform, a family of RISC (reduced instruction set computing) architectures for computer processors. Currently, processors based on the ARM platform are widely used in smartphones, tablets, smart wearable devices, and are being introduced into many other electronic devices. Therefore, developing efficient algorithms for such systems has attracted considerable attention. Many vision-based applications for embedded systems, including object tracking, have been already developed [6–8]. However, embedded systems have limited computing power and memory. This limitation prevents embedded systems from performing complex tasks, such as object tracking, within a reasonable execution time. For example, robust tracking algorithms based on high-complexity computing methods, like principal component analysis or image features such as SIFT (scale-invariant feature transform) and SURF (speeded-up robust features), are able to run in real-time only on modern powerful computers [9]. There is a trade-off between speed and reliability. Fast algorithms are less universal under certain conditions, and vice versa. For this reason, the development of algorithms which are both fast and reliable remains an ongoing problem.

In this research, particle filter is selected as the tracking framework among the common approaches for object tracking. Particle filter, a point tracking method, is used to perform estimation in state-space models, where the information of the state is acquired over time. It has been proven to produce high performance when applied to nonlinear and non-Gaussian estimation problems [10], hence particle filter is a robust solution for object tracking. This method approximates a posterior probability density of the state. In this case, it is the positions of the object in image sequence. The approximation is conducted by using point mass representations of probability densities, which are called particles. Usually contours, color features, or appearance models are used as particles when applying particle filter to object tracking [10–14]. A significant disadvantage of particle filter is computational complexity. The complexity increases as the area of the tracked region and the number of particles increase. When the dimensionality of the state space increases, the number of particles required for the sampling increases exponentially. A powerful computer is required for the purpose of producing results in a reasonable time. Fortunately, as a result of the rapid development of the semiconductor industry, embedded systems have become more powerful. This research presents a method for utilizing modern hardware architecture of embedded systems in an object tracking task. Parallel programming has been applied successfully in computer vision, showing its effectiveness in reducing the execution time of high-complexity tasks [15]. This is another reason for particle filter to be chosen, because parallel programming can be applied to increase computing speed while retaining tracking accuracy. In this paper, we introduce a parallel implementation of particle filter algorithm, for the purpose of enhancing the performance of the particle filter based object tracking method when deployed in embedded systems.

Methods

Particle filter

Tracking, including object tracking in computer vision, can be considered a discrete-time nonlinear filtering problem [16]. The aim is to estimate the state variables of a dynamic system by using noisy observations. Let t indicate the time step. Then the advancement of the state x_t of a discrete dynamic system is defined as

$$x_{t+1} = f_t(x_t, w_t), \tag{1}$$

where f_t is the transition function and w_t is system noise which has a known distribution. At each time step t, an observation z_t of the state x_t is acquired as

$$z_t = g_t(x_t, \tilde{w}_t), \tag{2}$$

where g_t is the measurement function and $\tilde{w}_t$ is measurement noise which also has a known distribution. Usually, the distribution of system noise and measurement noise is Gaussian, but in a nonlinear filtering problem, these distributions are more complex. At time k, the set of measurements $Z_k = \{z_t, t = 1, \ldots, k\}$ is available and the initial *p.d.f* (probability distribution function) $P(x_0)$ is given.

The estimation of state variables consists of two stages: prediction and update. In the prediction stage, given the *p.d.f* $P(x_{k-1}|Z_{k-1})$ at time $k-1$, the prediction $P(x_k|Z_{k-1})$ is calculated as

$$P(x_k|Z_{k-1}) = \int P(x_k|x_{k-1})P(x_{k-1}|Z_{k-1})\,\mathrm{d}\,x_{k-1}. \tag{3}$$

In the update stage, the prior *p.d.f* $P(x_k|Z_{k-1})$ at time k is updated using Bayes' rule

$$P(x_k|Z_k) = \frac{P(z_k|x_k)P(x_k|Z_{k-1})}{P(z_k|Z_{k-1})}, \tag{4}$$

where

$$P(z_k|Z_{k-1}) = \int P(z_k|x_k)P(x_k|Z_{k-1})\,\mathrm{d}\,x_k. \tag{5}$$

Particle filter is a method of solving this nonlinear estimation problem. In general, particle filter is an iterative algorithm, where each iteration consists of four stages: prediction, update, estimation, and resampling. First, the algorithm is initialized by generating a set of random particles, and the initial *p.d.f* $P(x_0)$ is estimated from this particle set. Let x_k^i denote the particle i, and w_k^i denote its corresponding noise, both at time k. Each particle has its own transition function and advances independently. The prediction stage produces an approximation of the a priori *p.d.f*

$$P(x_k) = \frac{1}{N}\sum_{i=0}^{N}\delta_{x_k^i}(x_k), \tag{6}$$

where δ_x denotes Dirac's measure and N is the number of particles. After prediction, the weight of each particle is updated using the obtained observations. Let ρ_k^i be the weight of particle i at time k that can be calculated as

$$\rho_k^i = \frac{P(z_k|x_k^i)}{\sum_{j=1}^{N}P(z_k|x_k^j)\rho_{k-1}^j}\rho_{k-1}^i. \tag{7}$$

Particles are weighted based on likelihood, i.e. particles which correspond to the most probable state will have a higher weight than others. The update function is defined as

$$P(x_k|Z_k) = \sum_{i=1}^{N}\rho_k^i\delta_{x_k^i}(x_k). \tag{8}$$

In the estimation stage, the estimator $\hat{x}_k$ is calculated by a weighted sum of particles as

$$\hat{x}_k = \sum_{i=1}^{N}\rho_k^i x_k^i. \tag{9}$$

The final stage of particle filter is resampling, also called particle redistribution. The purpose of this stage is to prevent degeneracy. Without resampling, the whole weight will be accumulated at a single particle after a few iterations, because each particle advances independently. Resampling is also the initialization for the next iteration. In this stage, a new set of particles is generated by redistributing all current particles based on their current weights, and each new particle is assigned new weight, $1/N$, after redistribution. However, resampling leads to another problem called sampling impoverishment. In this phenomenon, high-weight particles have a greater chance of being drawn multiple times during redistribution, whereas low-weight particles have a small chance of being drawn at all. This decreases the diversity of the particles after a few resampling steps. In the worst case scenario, all particles might be "merged" into a single particle. Sampling impoverishment decreases the performance of the tracking process drastically. Therefore, an appropriate particle redistribution method is required. A review of resampling methods for particle filter is provided in [17].

Color model

In tracking framework, the particle filter-based tracker requires a similarity measurement for the purpose of calculating the likelihood between particles and the target object. Before presenting our measurement, we will provide a brief introduction regarding visual features that are usually used in object tracking. As in [5], common visual features are color, edge, optical flow, and texture. For color features, the color of an object is affected mainly by two physical elements, the spectral power distribution of the illuminant and the surface properties of the object. In digital images and videos, colors are organized into specific sets called color spaces. This is an abstract mathematical model which simply describes the range of colors as tuples of numbers. There exist several color spaces for various purposes, and RGB (Red-Green-Blue) is the most commonly used color space [18]. However, the RGB space is not a perceptually uniform color space, in other words, small changes in RGB values may lead to large color differences perceived by the human eye. In contrast, L*u*v* and L*a*b* are perceptually uniform color spaces, while HSV (Hue-Saturation-Value) is an approximately uniform color space. However, these color spaces are sensitive to noise, which is a significant disadvantage because it is impossible to obtain noise-free data from image acquisition devices. Currently, there is no color space that is suitable for all vision-based applications. The choice of color space depends heavily on the goal of the application.

Edges are defined as strong changes in intensity values between object boundaries. Techniques used to identify these changes are called edge detectors. Edges are less sensitive to illumination changes than color features. This is an important property in object tracking because the algorithms are required to operate in various environments. The most popular edge detection technique is the Canny edge detector because of its simplicity and accuracy. Optical flow is the motion pattern of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (human eye or camera) and the scene. It is computed using the brightness constraint, which assumes brightness constancy of corresponding pixels in consecutive frames. Other than object tracking, optical flow is also used in motion estimation or video compression. For texture, it is a measure of the intensity variation of a surface which quantifies properties such as smoothness and regularity. Image texture gives us information about the spatial arrangement of color or intensities in an image or local regions of an image. This feature requires preprocessing procedures to extract texture information from images before being applied in desired algorithms. Texture features, like edge features, are also less sensitive to illumination changes.

Particle filter is a high-complexity algorithm; hence, a simple visual feature descriptor is required in order to prevent the increase of execution time of the whole tracking algorithm. Color distribution similarity is chosen because of its simplicity in calculation. We will combine particle filter and color information by integrating color distribution into particles. Each particle is defined as a rectangular region in the video frame and equal in size to the target object. Color distributions of particles are represented by three-dimensional RGB histograms. This color space is selected because of its robustness against noise and occlusion.

The histograms are calculated by $h(y_l)$, that assigns one of the M bins to a given color pixel at location y_l. In detail, we produce a histogram of a local region by discretizing the color pixels in the region into bins, and then count the number of pixels in each bin. For the discretization step, first we normalize the color values of pixels by dividing the value of each component by L. Then, we quantize the normalized values into m bins. L and m are the number of levels and bins for each component, respectively. These parameters must be the same for all three components. Let r, g, and b be the values of the Red, Green, Blue components, respectively, of the pixel at location y_l. Then, the equation $h(\cdot)$ for assigning the histogram bin is defined as

$$h(y_l) = \left\{ i, j, k \mid i = \left\lfloor \frac{r}{L} \times m \right\rfloor, j = \left\lfloor \frac{g}{L} \times m \right\rfloor, k = \left\lfloor \frac{b}{L} \times m \right\rfloor \right\}, \tag{10}$$

where i, j, and k are indexes of the chosen bin. For example, with $L = 256$ (8-bit RGB) and $m = 8$, the pixel at location y_0 with RGB values (45, 172, 103) will be assigned to the bin with index (1, 5, 3). The total number of bins is $M = 8 \times 8 \times 8 = 512$. Then, the number of pixels in each bin is used to construct the final histogram. Figure 1 shows an illustration for the output three-dimensional histogram from our calculation. The number of pixels in each bin is represented by size of the spheres.

After obtaining histograms of particles, the distance between the two color histograms is calculated using the Hellinger distance, which is also called Bhattacharyya distance because this was introduced by Anil Kumar Bhattacharya. Moreover, it is derived from the Bhattacharya coefficient. The Hellinger distance is chosen because of its effectiveness in object tracking [19]. First, given two discrete probability distributions $P = (p_1, \ldots, p_U)$ and $Q = (q_1, \ldots, q_U)$, the Bhattacharya coefficient is calculated as

$$B_C(P, Q) = \sum_{u=1}^{U} \sqrt{p_u q_u}. \tag{11}$$

The Bhattacharyya coefficient is a measurement of the amount of overlap between two distributions. Higher values of coefficient mean that the two distributions are closer to

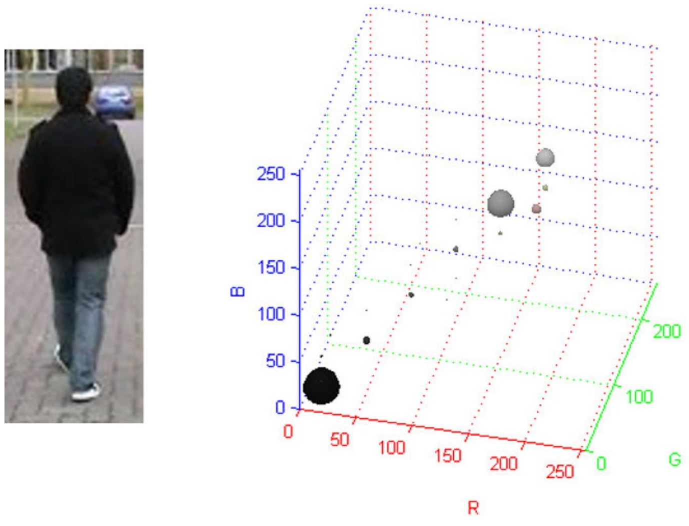

Fig. 1 3D histogram. Sample image region and its 3D histogram

each other, which leads to shorter distance. Therefore, the Hellinger distance is defined as

$$H_D(P, Q) = \sqrt{1 - B_C(P, Q)}. \tag{12}$$

When applied to calculate the distance between two color histogram H_1 and H_2, the formula for Hellinger distance is rewritten as

$$H_D(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H_1}\bar{H_2}M^2}} \sum_I \sqrt{H_1(I) \times H_2(I)}}, \tag{13}$$

where

$$\bar{H_1} = \frac{1}{M} \sum_I H_1(I),\ \bar{H_2} = \frac{1}{M} \sum_I H_2(I), \tag{14}$$

and I is the index of histogram bins, the term $(\sqrt{\bar{H_1}\bar{H_2}M^2})^{-1}$ is used for normalization. The proposed tracker employs the Hellinger distance to update the a priori distribution calculated by the particle filter.

Tracking framework

The particle filter in this research, which is developed to track movement of objects, is based on the Condensation algorithm [10]. The state vector $X_t = \{x_0, \ldots, x_t\}$ describes the state of the tracked object, and Z_t is the vector which stores all the observations of object movements $\{z_0, \ldots, z_t\}$ up to time t. The Condensation algorithm estimates the object state by calculating the conditional probability density $P(X_t|Z_t)$ using a nonlinear filter. $P(X_t|Z_t)$ represents the possible states of the tracked object based on previous states and measurements. Usually, the posterior density $P(X_t|Z_t)$ and the observation density $P(Z_t|X_t)$ are non-Gaussian, which increases the complexity of the tracking process.

In this tracking framework, let $S = \{(s_n, \pi_n) \mid n = 1 \ldots N\}$ denote the weighted particle set, where N is the number of particles. This particle set is used for approximating a probability distribution, which represents the object movement. For each particle, s is the location of the image region with its associated histogram, and π is the corresponding weight which obeys

$$\sum_{n=1}^{N} \pi_n = 1. \tag{15}$$

The movement of the tracked objects is described by a statistical model. In the tracking process, each particle is weighted depending on its likelihood with the observation of the target object. Then, N particles are drawn using resampling techniques. The tracked object is localized by the estimation of mean state, which is calculated at each time step by

$$\tilde{S} = \sum_{n=1}^{N} \pi_n s_n. \tag{16}$$

Because particle filter has the ability to model the uncertainty of object movements, it can provide a robust tracking framework. It can also consider multiple state hypotheses simultaneously [20]. On the other hand, particle filters are able to produce high accuracy predictions from previous observations, hence it can deal with short-time occlusions and sudden changes in object movement.

Parallel implementation

Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large and complex calculations can often be divided into smaller ones, which can then be solved simultaneously. Parallelization can utilize the multi-core architecture of modern embedded systems, in which all cores of the processor take part in the calculation process. Parallel implementation can decrease the processing time of these steps, which leads to higher performance in object tracking due to higher processed frames per second. However, parallelization has its own limitation. Generally in parallel computing, a complex task is first split into discrete parts that can be solved concurrently. Following this, these parts are distributed to several threads, and are then solved separately in each thread. An overall coordination mechanism is employed to control the status of the threads. After completing the computation, these threads will communicate with each other using this mechanism in order to produce final results. In some cases, the time required for communication is higher than that for solving the split tasks. The overall execution time of parallel implementation in these cases may be higher than that for sequential implementation. This phenomenon is called parallel slowdown.

In the particle filter algorithm, the most complex and time-consuming steps are resampling of the particles and calculating the likelihood between the particles and the target object. Another complex step is calculating the weighted mean of the particles, however implementing this step will lead to parallel slowdown due to data dependency. Therefore, we will use parallel programming to reduce the execution time of the resampling and likelihood calculation. The detailed algorithms for these steps are listed below.

Algorithm 1. Systematic resampling

```
Create new array newArr for storing output of resampling process
for $i = 1$ to (number of particle)
    $j \leftarrow 1$
    while (probabilities of particle($j$)) $<$ (likelihood $i$ with target object)
        $j \leftarrow j + 1$
    endwhile
    newArr($i$) $\leftarrow$ particle($j$)
endfor
```

Algorithm 2. Likelihood calculation

```
for $i = 1$ to (number of particle)
    Get image region of particle $i$
    Calculate likelihood between obtained image region and target object
endfor
```

Since there are no data dependencies between particles in these algorithms, parallel threads do not need to communicate with each other during processing. For example, in the resampling step, newArr(1), newArr(2), newArr(3), and newArr(4) can be calculated simultaneously using four threads, and hence reduce the execution time of the whole process while avoiding parallel slow down.

Experimental results

In this section, we present the performance of the object tracking method. Both versions of the particle filter algorithm, sequential and parallel, are implemented in C++ on a Linux operating system. OpenCV library is used for processing the video frames. For parallelization, we use the OpenMP library for the implementation of parallel particle filter. The program is deployed in embedded board Odroid U3, which has a quad-core processor running at 1.7 GHz and 2 gigabytes of memory. The videos used for testing are acquired from [21]. All videos has a resolution of 320×240. In the next sections, we will show the efficiency of the particle filter object tracker and the comparison of performance between sequential and parallel implementation.

Tracking accuracy

The tracking process is shown in Figs. 2 and 3. In both figures, each column shows the tracked target (top) and the locations of particles (bottom) of the same frame. The color-based particle filter is configured to be executed with 300 particles, and the histograms are calculated in the RGB color space using 8 bins for each component. In Fig. 2, the cup was moved in front of a background with complex color patterns. However, the proposed object tracker can localize the cup with high accuracy. Figure 3 shows that the object tracker can deal with occlusions, as it did not lose trace of the tracked person after the occlusion occurred. From the experimental results, the color-based particle filter has been proved to yield good performance when applied in object tracking.

Computational performance

In this section we will consider the computational performance when processing the video shown in Fig. 3. Figures 4, 5, and 6 show the comparison between two implementations during the execution time of shuffling, resampling, and likelihood calculation steps when processing the first 50 frames of the video. The parallel implementation is configured to be executed using four threads. As shown in the figures, the parallel

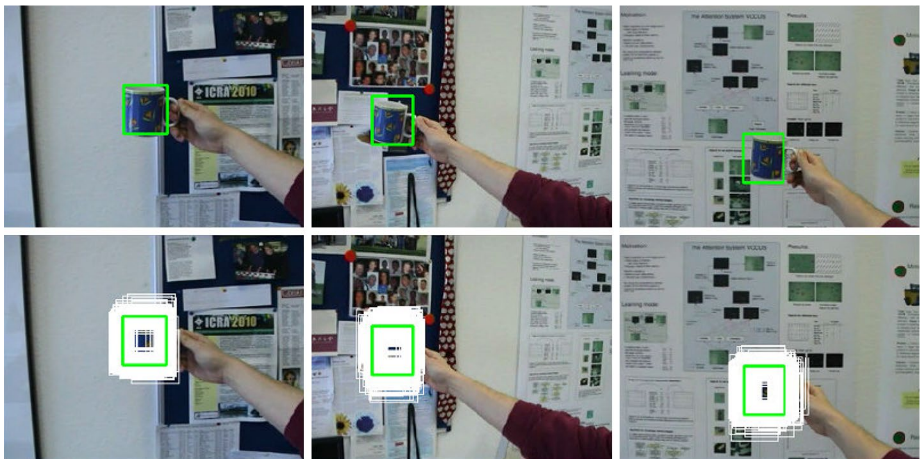

Fig. 2 Object tracking. Tracking moving cup with complex background

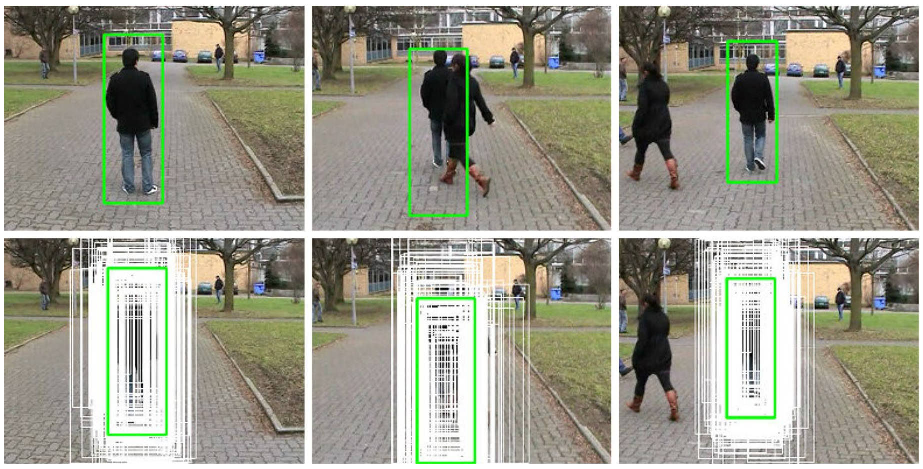

Fig. 3 Person tracking. Track walking person with occlusions

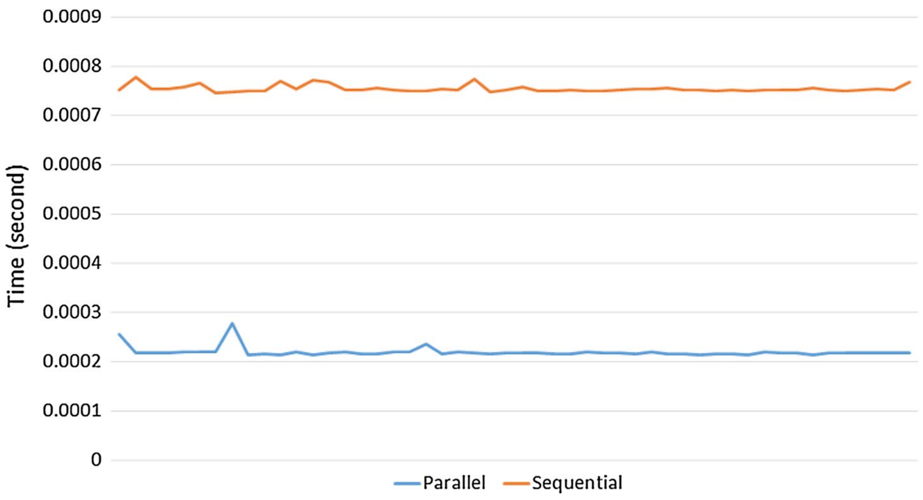

Fig. 4 Particle shuffling. Execution time comparison in particle shuffling step

implementation of particle filter is at least two times faster than the non-parallel implementation in all steps.

Figure 7 shows the execution time of both implementations in processing the whole video. With 300 particles, parallel implementation took approximately 62 s to process 1017 frames (16 frames per second), while non-parallel implementation took 92 s (11 frames per second). The overall performance is increased by 50% when applying parallel programming. However, this improvement does not scale with number of threads used. In other words, using four threads does not mean the performance will be four times better than using one thread, i.e. sequential programming, because some minor steps in the algorithm are processed sequentially. Unfortunately, parallelizing these steps will lead to parallel slowdown.

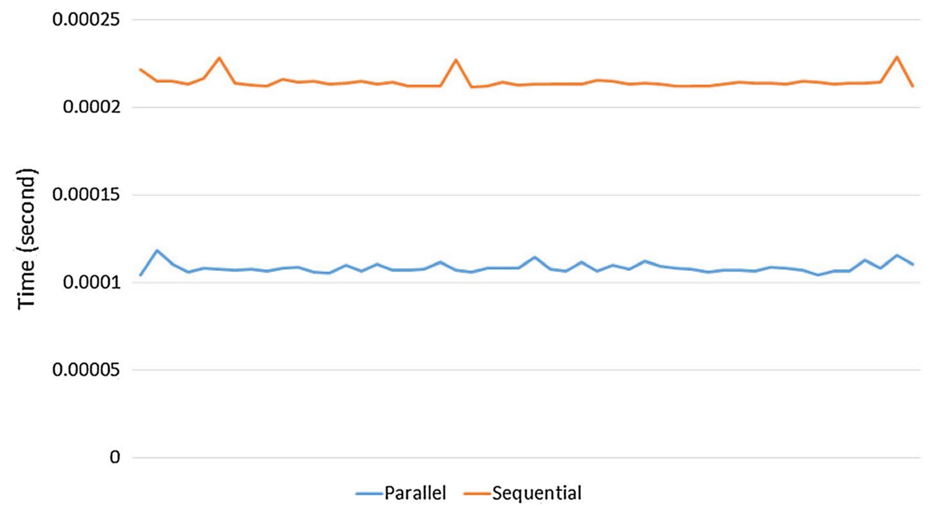

Fig. 5 Particle resampling. Execution time comparison in resampling step

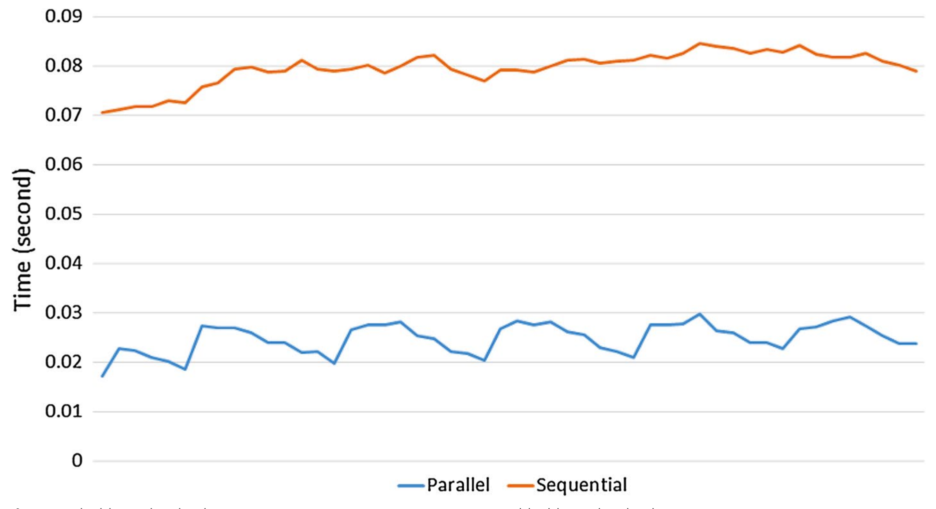

Fig. 6 Likelihood calculation. Execution time comparison in likelihood calculation step

Conclusion

Recently, embedded systems have become popular with many applications. Developing efficient algorithms for such systems has attracted considerable attention. This research aims to increase the performance of the particle filter-based object tracking method when the algorithm is deployed in embedded systems by using parallel programming. A significant limitation of embedded systems is their computing power, hence it is difficult to use embedded systems for high-complexity computation. After several years of development, embedded systems have become more powerful. Currently, modern systems have high memory and multi-core architecture. However, the application must also utilize this advantage. In this research, we have implemented a parallel particle filter for object tracking in order to utilize the multi-core architecture. The proposed object tracker has shown its efficiency in different tests. The program is also deployed in

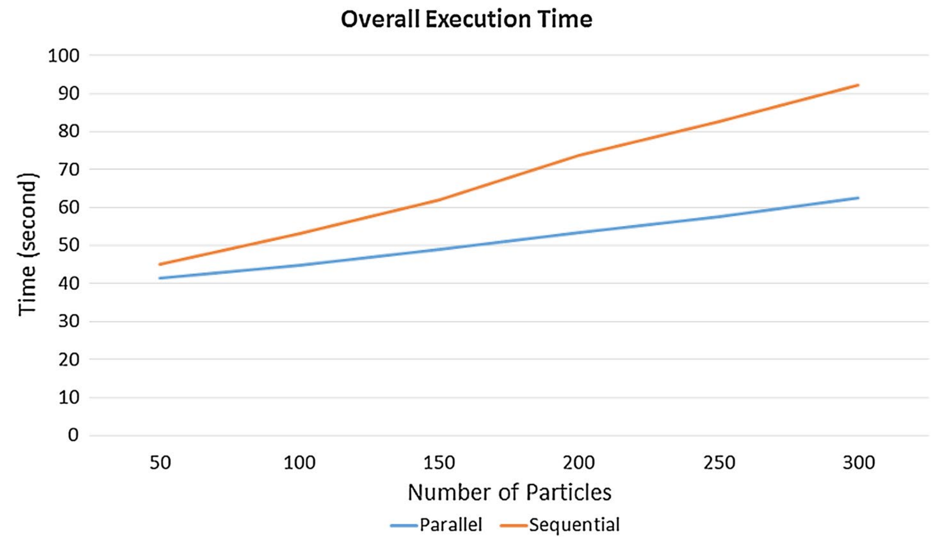

Fig. 7 Performance comparison. Overall execution time for processing whole video

Odroid, an embedded board with a four-core processor. With 300 particles, the object tracker with sequential implementation can process 11 frames per second, whereas parallel implementation can process 16 frames per second while retaining the tracking accuracy. The experimental results show that multi-core embedded systems can produce higher performance if the hardware is used at its maximum potential. However, the color model is not effective if the illumination of the environment changes. The proposed object tracking method also loses track of the object if there are other objects with similar color pattern. For future research, we will combine multiple features to increase the tracking accuracy. On the other hand, the tracking method must be optimized for the purpose of deploying in embedded systems.

Authors' contributions

MTNT designed and developed the algorithm and took suggestions as and when necessary from Professor SK. Both authors read and approved the final manuscript.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057518).

Competing interests

The authors declare that they have no competing interests.

Received: 17 August 2016 Accepted: 8 December 2016
Published online: 09 January 2017

References

1. Kenan M, Fei H, Xiangmo Z (2016) Multiple vehicle detection and tracking in highway traffic surveillance video based on sift feature matching. J Inf Process Syst 12:183–195
2. Juhyun L, Hanbyul C, Kicheon H (2015) A fainting condition detection system using thermal imaging cameras based object tracking algorithm. J Converg 6:1–15
3. Bostanci E, Kanwal N, Clark AF (2015) Augmented reality applications for cultural heritage using kinect. Hum Cent Comput Inf Sci 5:1–18
4. Smeulders AWM, Chu DM, Cucchiara R, Calderara S, Dehghan A, Shah M (2014) Visual tracking: an experimental survey. IEEE Trans Pattern Anal Mach Intell 36:1442–1468
5. Yilmaz A, Javed O, Shah M (2006) Object tracking: a survey. ACM Comput Surv 38:1–45

6. Fiack L, Cuperlier N, Miramond B (2015) Embedded and real-time architecture for bio-inspired vision-based robot navigation. J Real Time Image Process 10:699–722
7. Hammoudi K, Benhabiles H, Kasraoui M, Ajam N, Dornaika F, Radhakrishnan K, Bandi K, Cai Q, Liu S (2015) Developing vision-based and cooperative vehicular embedded systems for enhancing road monitoring services. Proced Comp Sci 52:389–395
8. Lu F, Lee S, Kumar Satzoda R, Trivedi M (2016) Embedded computing framework for vision-based real-time surround threat analysis and driver assistance. In: The IEEE conference on computer vision and pattern recognition (CVPR) workshops, vol 1, pp 83–91
9. Varfolomieiev A, Lysenko O (2016) An improved algorithm of median flow for visual object tracking and its implementation on arm platform. J Real Time Image Process 11:527–534
10. Isard M, Blake A (1998) Condensation-conditional density propagation for visual tracking. Int J Comput Vis 29:5–28
11. Isard M, Blake A (1998) In: Burkhardt H, Neumann B (eds) Icondensation: unifying low-level and high-level tracking in a stochastic framework. Springer, Berlin, pp 893–908
12. MacCormick J, Blake A (2000) A probabilistic exclusion principle for tracking multiple objects. Int J Comput Vis 39:57–71
13. Wu Y, Huang TS (2004) Robust visual tracking by integrating multiple cues based on co-inference learning. Int J Comput Vis 58:55–71
14. Khan Z, Balch T, Dellaert F (2004) A rao-blackwellized particle filter for eigentracking. In: Proceedings of the 2004 IEEE computer society conference on computer vision and pattern recognition, vol 2. CVPR, pp 980–986
15. Moulkheir Naoui SM, Belalem G (2016) Feasibility study of a distributed and parallel environment for implementing the standard version of AAM model. J Inf Process Syst 12:149–168
16. Stepanov OA, Vasil'ev VA (2016) Cramér-rao lower bound in nonlinear filtering problems under noises and measurement errors dependent on estimated parameters. Autom Remote Control 77:81–105
17. Li T, Bolic M, Djuric PM (2015) Resampling methods for particle filtering: classification, implementation, and strategies. IEEE Signal Process Mag 32:70–86
18. Barthélemy Q, Larue A, Mars JI (2015) Color sparse representations for image processing: review, models, and prospects. IEEE Trans Image Process 24:3978–3989
19. Vojir T, Noskova J, Matas J (2014) Robust scale-adaptive mean-shift for tracking. Pattern Recognit Lett 49:250–258
20. Nummiaro K, Koller-Meier E, Van Gool L (2002) In: Van Gool L (ed) Object tracking with an adaptive color-based particle filter. Springer, Berlin, pp 353–360
21. Tracking Dataset. http://cmp.felk.cvut.cz/~vojirtom/dataset/