



Optimisation of mobile intelligent terminal data pre-processing methods for crowd sensing

Min Huang¹ ✉, Yuefan Zeng¹, Lina Chen¹, Bo Sun²

¹School of Software Engineering, South China University of Technology, Guangzhou, People's Republic of China

²Research Institute of International Services Outsourcing, Guangdong University of Foreign Studies, Guangzhou, People's Republic of China

✉ E-mail: minh@scut.edu.cn

ISSN 2468-2322

Received on 5th December 2017

Revised on 12th January 2018

Accepted on 19th January 2018

doi: 10.1049/trit.2018.0013

www.ietdl.org

Abstract: Sensor data pre-processing is an essential phase of crowd sensing application. Existing studies do not effectively solve the problem, and there still exist various sensor data pre-processing optimisation problems at the acquisition end in crowd-sensing process. This study presents an improved sliding average method to achieve data compression and reduce the time complexity by using a dynamic window with improved processing time. Through adopting locally sorting and gradient change of the filter window, an improved extremum median filtering method is proposed to relieve the time-consuming problem when denoising high pixel images. A transmission strategy for optimisation is also proposed, in which only the demarcation points of each group of data and the data points with large differences when comparing with the demarcation points are recorded. This strategy reduces the storage pressure and the amount of data transmission of mobile terminal and improves the efficiency of data transmission. The experimental results show that their methods have higher speed and lower cost, and thus they can run better in crowd-sensing environment.

1 Introduction

In crowd sensing, ordinary users of mobile devices, such as mobile phones and tablet computers, can act as basic sensing units, and they consciously or unconsciously participate in some large scale and complex social sensing tasks by collecting and sharing sensing data [1]. Due to the ubiquitous usage of mobile intelligent terminal devices with a variety of embedded sensors, the computing and sensing abilities of these intelligent devices have greatly improved, which makes crowd-sensing technologies develop quickly and be applied to many areas, such as intelligent transportation [2, 3], environmental monitoring [4, 5] and community information sharing [6, 7].

However, because the hardware processing performance is limited, the storage capacity is insufficient, and the external sensing environment is complicated. There are still some difficulties and key problems need to be solved in the terminal sensing data pre-processing of intelligent mobile devices before they can be used in various crowd-sensing applications.

Firstly, in the collecting process of sensing data, the collected data inevitably include some errors owing to limitations of devices and environment. Thus, it is necessary to filter these original sensing data in a pre-processing stage. However, many fragmental data items can be accumulated in a short time for sensing data refreshing quickly. If these sensing data items are not filtered out and are directly recorded without distinction, there will be much more pressure on mobile devices for pre-processing.

Secondly, it is hard to support the data pre-processing algorithms with high time and space complexity in mobile phones because of their hardware limitation. Thus, most existing crowd-sensing applications directly use the original sensing data, or process them by some simple algorithms lacking pertinence, which reduces effectiveness of sensor data pre-processing and cannot sufficiently adapt to the particular environment requirements of mobile terminals.

Thirdly, the limited storage capacity makes mobile terminals unable to store the quickly incoming sensing data in time, and thus these data have to be uploaded to the server to free up the space for next coming sensing data. Therefore, how to transmit

continuously generated sensing data with less loss in the quantity of original information of sensing data is also an important problem for pre-processing stage in crowd-sensing applications.

Aiming at the aforementioned problems, this paper focuses on conducting optimising studies on the pre-processing process of sensing data in crowd-sensing applications, such as information collection, filtering, compression and transmission, by which to improve the whole efficiencies of crowd-sensing application systems.

2 Analysis of related researching works

The front-end pre-processing methods for sensing data and their categories used in the existing crowd-sensing applications are introduced in this section.

2.1 Data pre-processing methods

In the category of numerical sensing data pre-processing, Han and Li [8] analysed error sources of acceleration sensors and characteristics of sensors themselves. The low-pass filters are used to correct bias errors and eliminate the influences of gravity acceleration on acceleration data. However, it does not consider the simplification and compression for acceleration-based data that refreshes quickly. Ji *et al.* [9] studied the intelligent terminal sensor data processing in real time by utilising a sliding average method and an interpolation normalisation method. Since this processing algorithm has a lower complexity and has less pressure on devices, it has some practical meaning to data processing on mobile devices. However, the sensing data are not compressed and the amount of data will become larger because of the use of interpolation. For viscosity of speed data, Wang and Li [10] studied a method to filter speed sensor data and remove outliers by taking advantage of RBF neural network. In this study, the experiment environment is based on the computers in factories. In contrast, the computing power of mobile devices is much lower. Therefore, it is unsuitable for directly adopting this kind of

pre-processing methods which need certain amount of data for data modelling. Qin *et al.* [11] utilised the theory of information entropy to compute data uncertainty and clear uncertainty data which are outside the confidence interval. Hu *et al.* [12] proposed a method to detect outliers of sensor data by taking advantage of BP neural network and linear neural network. However, both of the two adopted methods need certain amount of data preparation and data modelling, and training. Thus, it is unsuitable for sensor data, which refresh rapidly and have a high demand for real-time processing.

In the category of image denoising, the study in [13] amends expanding conditions of filtering windows and optimises the adaptive median filtering method to preserve more image details by introducing minimum difference of grey value and improving the method of noise point detection. Pan *et al.* [14] proposed a method to obtain the change range of filtering window size by utilising image pixels' polluted level and reduce the sorting complexity of median filtering by utilising a divide and conquer algorithm. The study in [15] first utilises grey value to detect noise points, and then it processes the selected noise points through linear prediction algorithm. In [16], it denoises images by combining standard median filtering and weighted median filtering, and decides to use which method according to gray values of pixels. The approach makes the filtering more targeted. In [17], it makes noise point judging conditions of extremum median filtering more detailed. It combines wavelet denoising and determines not only extremum of grey values but also edges, which reduces the misjudgements of noise points. In [18], it optimises the sorting algorithm when median filtering is searching the median within the range and reduces the algorithm complexity by making local sort for the pixel value matrix only.

2.2 Sensing data transmission methods

In [19], it studies the packing technology when sending a set of data, which recovers the lost parts forwards through the received ones rather than resending packets. In [20], it summarises all kinds of low-cost data transmission methods and protocols of wireless sensor network and discusses several data transmission details in the sensor network, such as packing and congestion control. However, the discussion does not consider the specific sensing data. In [21], it transmits data by means of 'reference value + change value'. It only transmits values with small change, while ignoring data packets, which changes less in data. If the server does not receive the packet in the given time, the data will be replaced by the previous one. Therefore, it reduces the quantity of packets that the server needs to receive, but it cannot be directly applied to the sensors whose sampling time is not even and sampling frequency is high.

In summary, applying crowd-sensing applications to data pre-processing is mainly facing the following five problems:

(1) *Insufficient demand analysis of sensing data pre-processing:* Most existing studies directly apply optimised data pre-processing algorithms to certain sensing data. They do not specifically analyse numerical features and pre-processing demands of sensing data, which makes the pre-processing methods lack pertinence and leads to failing in fulfilling pre-processing demands of sensing data. The results of inadequate analysis are mainly presented in the following problems (2)–(4).

(2) *Pre-processing of the massive data:* All the original data collected by different kinds of sensors include errors and outliers. Thus, they must be filtered before storing. Varieties of mobile intelligent terminals and high sampling frequency mean that the amount of data need to be pre-processed will be very large. Thus, how to filter massive fragmental sensing data is a quite important problem that crowd-sensing data pre-processing needs to solve.

(3) *Compression and transmission of massive data:* Another problem that high sampling frequency of sensors brings about is data redundancy. It is caused by mass accumulation of sensing data collected at the same time, which makes the obtained data

files bloated. For alleviating storage and transmission pressure on mobile intelligent terminals, we need to simplify and compress sensing data without too much loss in the information amount of original data.

(4) *Denoising of high-pixel images:* With the continuous progression in image sensing technologies, the image pixel value collected by mobile intelligent terminals is increasing and the execution time of the image denoising algorithms in pixels is also prolonged. In order to improve the operational efficiency of crowd-sensing applications using image sensing data, it is necessary to optimise existing image denoising algorithms for speeding up processing a single pixel [22].

(5) *Adaption of pre-processing methods to mobile intelligent terminals:* The storage space and computing power on mobile intelligent terminals are limited. Thus, the complexity of the adopted pre-processing method cannot be too high, and the method must be optimised and improved to avoid too much pressure on mobile intelligent terminals, while it can ensure pre-processing effects.

From the aforementioned analyses we can see, although existing studies have contributed a lot to solving some key problems of data pre-processing of crowd-sensing applications, there are still many inadequate analyses in the features of sensing data and processing demands of crowd-sensing applications. There are still much room for improving the efficiency of pre-processing of massive sensing data and adaptability of mobile intelligent terminals. Therefore, this paper makes some optimisations for the pre-processing process considering certain factors such as the characteristics of various sensing data, the characteristics of mobile intelligent terminals, and the data processing requirements for different crowd-sensing applications in order to achieve better effects.

3 Sensing data preprocessing improvement

In this section, we describe three improved approaches for sensing data preprocessing proposed in this paper, respectively. Description of every approach consists of three parts. First, we analyse data features and processing demands of related sensing data, and then we summarise the limitations of existing algorithms and the limitations when they are applied to sensing data. Finally, we propose the specific improved approach.

3.1 Double-cache sliding average method of using time interval windows and limiting smooth times

The algorithm in this section aims at numerical sensing data. All the data obtained by numerical sensors are specific values. Numerical sensors have the largest number and the most varieties in all sensors embedded in mobile intelligent terminals and are applied to many fields, such as temperature monitoring, GPS positioning, and game operation assistance.

3.1.1 Numerical sensing data features and preprocessing demands:

The sampling frequency of numerical sensors is very high. When several numerical sensors work at the same time, the number of data per second to read can be in the range of dozens or even hundreds, and the data refresh rapidly. Thus, the requirements for real time and stability of data processing are high. It is necessary to find out outliers from currently obtained data instantly and write them into files after filtering without any influence on the ensuing data reading and processing.

In addition, it is unnecessary for users to record overly intensive data points in daily crowd-sensing applications, because it may result in a large accumulation of data expressing environmental situations at the same time. A more sensitive sensor may get a piece of data every 10 ms on average at the peak of sampling, which has no influence on users' intuitive feelings but makes data files too bloated when writing data files for recording. Thus, it is

necessary for numerical sensors to standardise the maximum number of data obtained in a unit time and simplify redundant parts to achieve the compression of data without damaging information quantity.

3.1.2 Existing methods and their limitations: The sliding average method is a classical data processing algorithm. Owing to its outstanding properties, it is still used frequently in today's actual data processing. The average method has a simple algorithm with small account of computing and is good at processing data in real-time rapidly. These advantages meet requirements of crowd-sensing applications and environment required for this paper, and thus it can be used to preprocess numerical data that refresh rapidly and have features of the linear change.

The basic idea of the sliding average method is as follows. In an appropriate interval, non-smooth data are regarded as smooth, and then locally average them to reduce the effects of errors. For a set of data with a length of N , keeping locally averaging from the beginning to the end can achieve smoothing of data and can filter out some errors [23]. For instance, assume that the data in an interval with a length of m are smooth, all the data in this interval can be replaced by the average of this interval. The average is the reading value after filtering out errors and noise, which can be regarded as the exact value.

The existing sliding average method has three major problems when applied to the preprocessing of numerical sensing data:

(1) *Poor processing effects of massive data and data redundancy:* The existing sliding average method is to indiscriminately smooth for all the data points. Suppose that the five-point average method is utilised. The data dimension is m and the cache size is n . Then, most data points in the cache must participate in smooth processing for five times, and the time complexity for a round of processing all the data in the cache is $O(mn)$. This algorithm complexity does not appear to be high, but if the processed data are collected by numerical sensors with a high sampling frequency, it can easily lead to excessive computation and collapse of data collection programs. Furthermore, in the previous section, it has been shown that a high sampling frequency of numerical sensors may cause the problem of data accumulation in a unit time, resulting in excessive space usage of data and poor readability. Unfortunately, the existing sliding average method cannot solve these problems.

(2) *Inapplicability to data with irregular time intervals:* The ideal environment for using the sliding average method is a set of data points with regular time intervals. However, in the data collection of numerical sensors, the time distance between data points is variable due to the data collection mechanism of 'only acquiring the change value'. For example, sometimes, the interval of collection time between two adjacent data points may be several seconds or even more than 1 min, which is detrimental to effects of the sliding average method. Since the data points with overly long interval may not meet the basic idea of sliding average method, 'data variation within a certain time window could be seen as smooth'. For tackling this problem, the study in [11] utilises the linear interpolation method that supplements the estimated value at the time of no reading to make time intervals more regular. However, the side effect of doing so is to further increase the amount of data points, making the problem (1) more severe.

(3) *Error deletion when clearing the cache:* Since numeric sensing data refreshes rapidly, it is likely to mistakenly delete the newly added data points when it thinks the number of data points in the cache reaches the threshold value and is going to clear the cache. In general, thread locking or semaphore can be used to solve this problem, that is it does not allow new data to join in when clearing the cache. However, this method can lead to the loss of new rapidly refreshed data.

Therefore, in order to relieve the pressure of calculation, to achieve the compression of data volume, and to avoid the weakening of filtering effects caused by the uncertainty of sampling frequency,

this paper proposes an improved solution that utilises a time window for the sliding average processing and limits maximum smooth times. At the same time, double-cache mechanism is adopted to prevent erroneous deletion when it is clearing the cache.

3.1.3 Improved solutions: In order to overcome these shortcomings, we propose two solutions described as follows:

The time interval window and the limitation of maximum smooth times: Generally speaking, the window size of the sliding average method is fixed, and it only moves the distance of one data point each time. The improving idea of this paper is to make the window size variable and to make it as long as the data section of the time interval whose current length is one second. Owing to choosing a short time slot of 1 and according to the sliding average method's basic idea, the data in this time slot could be seen as smooth. In addition, the improved method limits smooth processing times in a single time interval as well, which makes n values preserved at most after smoothing in each time interval. For example, if there are m data points in the current time interval, the corresponding window size in this time interval is m . If $m \leq n$, there is no need for processing. If not, the interval will be divided into n child data sections. The previous $n-1$ sections contain w data points and $w = \lfloor m/n \rfloor$. The last section includes $w_n = m - w(n-1)$ data points. When the data are preprocessed, we build the average of all data points in every child data section and there are n times in total for averaging. As a result, m data points accumulated in a short time can be simplified as n representative data points at most. Each time the window moves, the sliding distance of the window is no longer a fixed value, instead, the window size adjusts itself. In this example, the window in the left margin should move the distance of m data points to the right when it is going to process the data in next time interval. In this way, the window is equal to gap between time intervals rather than progressively move in the existing algorithm.

Assume that the first data point of a time window is y_1 . Sensor reading at time t is y_t and the extra value is f_t . Let $h = \lceil w/2 \rceil$, then the general formula in a time window of the sliding average method introduced time interval is as below:

$$f_{h+(k-1)w} = \frac{1}{w} \sum_{i=(k-1)w+1}^{kw} y_i, \quad (1)$$

$$k = 1, 2, \dots, n-1$$

Specially, let $h_n = \lceil w_n/2 \rceil$, for the last (the n th) time slot in the time window, there is:

$$f_{h_n+(n-1)w} = \frac{1}{w} \sum_{i=(n-1)w+1}^m y_i, \quad (2)$$

After this improving idea is applied, the time complexity of the improved method is still $O(mn)$. However, because the total amount of data points n is reduced significantly, the algorithm execution time is clearly reduced in the actual operation. The essence of this optimisation is to reduce the level of data details in exchange for the significant increase of data preprocessing speed. As for whether it worth losing the amount of data information to increase the speed, the later experiments will prove and justify it.

In the aspect of weight, if the weighted average is used and because of irregular time intervals between data points, it will be hard to define a rational method for getting the weight. If we use a dynamic weight according to the time interval, the situation that some outliers are assigned to the maximum weight may occur, leading to the exception of the final weighted average value. This is risky, and therefore, the method used in this paper is equally weighted average. Namely, the weight of each data point is the same.

In the aspect of data structure, it obtains the current time when collecting data (accurate to seconds) and inserts the data nodes with the same time into a same linked list. After all the nodes are

inserted, we add the linked list into the cache that is a linked list as well.

The optimised sliding average method is shown as the flow chart in Fig. 1.

A detailed description of each step is given as follows:

- Find the linked list storing all data points of next time slot from the cache.
- According to the number of data points at the current time point linked list, determine the size of the processing window m , and then obtain the size of each child-window w and wn by calculation.
- Use (1) and (2) to obtain the equally weighted average of data points in each time segment. Delete all the origin data and use the obtained n averages to represent the data in this time interval.
- Check whether there are unprocessed linked lists. If yes, go back to step (a). If not, terminate the process.

Double-cache mechanism: At each collection of data, determine whether the data points in the cache have reached the threshold. If the threshold has been reached and the data in the current time point have been recorded, then switch to another empty cache and continue to write. Meanwhile, clear the full cache. The operations of writing and clearing are done by different threads. The execution flow chart of the double-cache mechanism is given in Fig. 2.

A detailed description of each step is given as follows:

- Sensors collect data.
- Determine whether the cache size reaches the threshold and whether the data in the current time are recorded. If both are yes, go to step (c), otherwise return to step (a).
- Switch to the other cache to write and clear the full cache.
- If continue to collect data, go back to step (a), otherwise terminate the process.

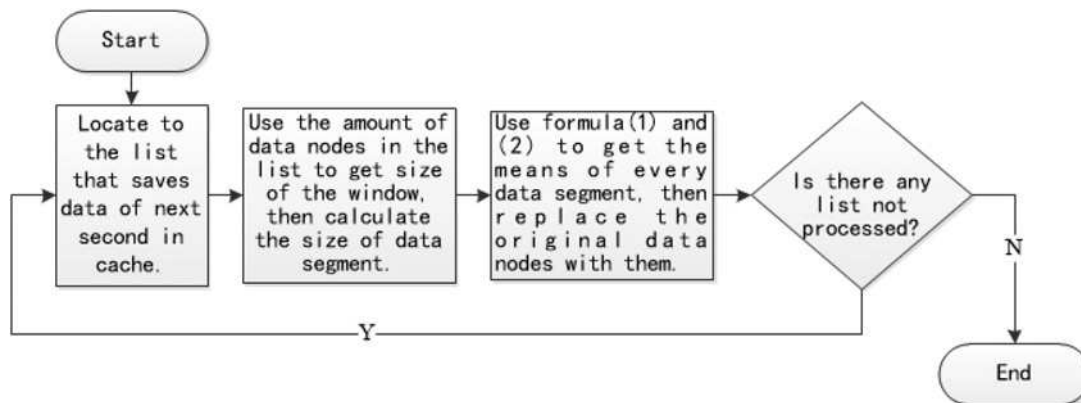


Fig. 1 Flow chart of the improved average method

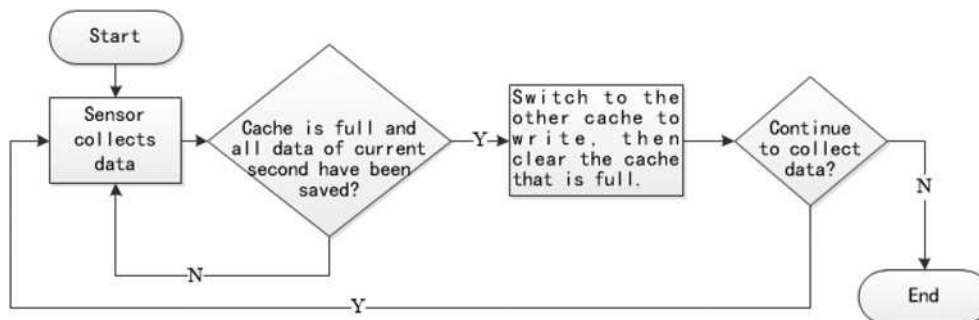


Fig. 2 Flow chart of cache switch process

In conclusion, by taking the advantage of time interval windows and double-cache mechanism, the improved method avoids situations of smoothing the data points with overly large time span and mistakenly deleting the new data points, which improves the accuracy of data processing. By limiting smooth times, the method effectively reduces time complexity.

3.2 Extremum median filtering method of improved sorting and filtering window movement

It is recommended that standardised fonts such as Times New Roman and Arial are used with a font size no smaller than 10 pt.

3.2.1 Image-based sensing data features and preprocessing demands: The sampling frequency of image-based sensing data is low and irregular. When the image sensor is working, because of external wave interference, equipment materials and illumination etc., the finally obtained image data will inevitably contain some random, discrete and isolate pixels that are called noises. They are equal to random errors of image data. Since the noise pixels are very different with the adjacent pixels, it will affect the image quality in some way. Thus, it is necessary to denoise the images obtained by sensors.

3.2.2 Existing methods and their limitations: The median filtering method is a classic image denoising algorithm. It can avoid making images fuzzy after denoising them, and the complexity is quite low. Thus, it is suitable to be used in mobile devices.

The idea of median filtering is similar to the sliding average method, but it still has some differences. For each pixel (x, y) in a black-white image with the size of $N \times M$, take a square block centred on the pixel whose length and width are both m , where m is generally odd, for facilitating the subsequent operation. The grey values of each pixel in the block are sorted and the

intermediate value is taken as the pixel value of point (x, y) [24]

$$\begin{aligned} f(x, y) = \text{mid}(f(x-n+1, y-n+1), f(x-n, y-n+1), \dots, \\ f(x+n-1, y+n-1))n < x < N-n, \\ n < y < N-n \quad \text{where } 2n+1 = m \end{aligned} \quad (3)$$

The images obtained in mobile sides mostly are colour images. When colour images are processed, the idea of the median filtering is still similar, but more complicated, because the grey values of colour images cannot be directly obtained. The grey value needs to be computed through formulas after obtaining the RGB colour components. There are many formulas for computing the grey value through RGB value, and the adopted method in this paper is the floating-point algorithm:

$$\text{grey} = R \times 0.3 + G \times 0.59 + B \times 0.11 \quad (4)$$

In addition, the calculations for the final pixel values of point (x, y) are different. Instead of directly using the median of grey value in the filtering window, we first calculate the median of three channels R, G, B pixel values of each pixel in the filtering window according to (3) and (4), and then combine with the transparency (Alpha value) to call the class colour method to get the final pixel value.

Compared with the traditional median filtering method, the extremum median method reduces some calculations because only the selected noise points are going to be calculated in the final pixel values. However, it still has much room for improvement.

Optimisation aimed at high pixel photos: The cameras' accuracy of existing mobile intelligent terminals is increasingly improving. The pixels of photos taken by daily mobile devices are up to millions, which is a big challenge for traditional image denoising algorithms that process images in pixels.

Optimisation of extremum and median calculations in the filtering window: The traditional method of extremum median filtering is to sort the grey values of all the pixels in the filtering window when searching for the extremum of grey values in the filtering window. Then, it obtains the maximum and minimum in the ordered sequence of grey values. However, it does not require all the grey values are ordered in this step. Only the extremum is required. Thus, partial sort can be taken into account. Similarly, it is unnecessary to sort all the RGB values when searching for the median in the filtering window. In [18], it studies the partial sort of grey value matrix in some way, but it does not combine the extremum, and it only considers the situation of simple black-white images. It does not discuss the optimisation of colour image denoising. Thus, it is still possible to further optimise the sorting method.

Assignment cost of filtering window movement: The traditional filtering window movement is to use a two-layer for loop, so that the filtering window traverses the image from left to right in each row. In this process, each movement will re-initialise the pixel information array stored in the filtering window, which is an expensive process. There is some related work optimising the array assignment during the process of filtering window movement. However, the descriptions of assignment processes are not specific. They do not take into account the situation when the filtering window moves to the image edge and do not discuss the selection of the window movement direction.

According to the aforementioned limitations, we optimised the traditional extremum median filtering method in two aspects, the sorting in the filtering window and the move of filtering window.

3.2.3 Ideas for improvement: In this paper, the optimising solution of the extremum median filtering method is proposed from the view of sorting. The efficiency of the algorithm is further improved by reducing the time complexity of searching regional extremum and median. The overall implementation process of the algorithm is still the same as described in Section 3.2.2. The optimising solution includes the following two parts:

Improved sorting in the filtering window: Take the filtering window with the size of 3×3 as an example. The traditional sorting method is to uniformly sort all the grey values of the 9 pixels, and then search the maximum, the minimum and the median in the ordered grey value sequence. There are about 30 comparisons required in the case of using the quick sort whose time complexity is $O(n^2)$.

The sorting steps in the filtering window after improvement are as follows:

- Sort the values in each row in ascending order.
- Sort each row in ascending order of first column values. The obtained sequence is not completely ordered, but the value at the upper-left corner must be the minimum.
- Sort each row in ascending order of the third column values, and then the value at the lower-right corner is the maximum.
- Sort each row in ascending order of the second column values, and the value at the matrix centre may not be the median. It is easy to know that the first two values of the first row are, respectively, smaller than 6 numbers and 5 numbers, and it is impossible to be the median. Similarly, the last 2 values of the third row cannot be the median. At the moment, if the remaining five values are sorted in ascending order, the median of the five values is the median of the window. However, due to the high requirement for the order of arrays in the next optimisation step and the demand of reducing the algorithm complexity on mobile sides, it is unnecessary to acquire the extra median, and thus the median will be replaced by the value of the matrix centre.

When getting the extremum of grey values, it only needs to execute the step (a) to step (c) in Fig. 3. When getting the median of RGB values, it only needs to execute the step (a) and step (d).

The diagrams of the entire sorting process are shown in Fig. 3. The numbers in the diagrams indicate the size of the value, such as 1 for the minimum, 9 for the maximum.

In the above process, step (a) requires nine comparisons, and (b), (c) and (d), respectively, requires three comparisons. Thus, it only requires 18 comparisons to obtain the extremum and median. Compared to the traditional sorting method, it makes significant improvement. Specifically, for grey values, it only needs to acquire the extremum and does not need to take the step (d). Thus, there are only $9+3+3=15$ comparisons. For the pixel values of RGB three channels, it only needs to acquire the median and does not need to take step (b) and step (c). Thus, there are only $3 \times (9+3)=36$ comparisons. If the grey value of the current pixel is the extremum in the filtering window, total ordering needs to be done for the four 3×3 filtering windows when the traditional method is used, namely $30 \times 4=120$ comparisons on average.

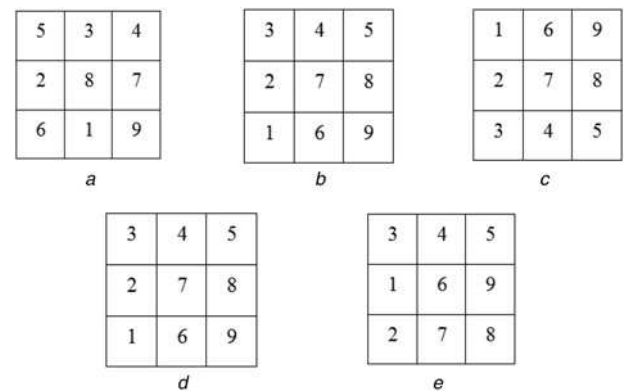


Fig. 3 Diagrammatic sketch of sorting process

- Beginning
- After phase a
- After phase b
- After phase c
- After phase d

However, there are only $15 + 36 = 51$ comparisons need to be done in this paper's method which reduces 69 comparisons, compared to the traditional method, and can greatly improve the denoising performance.

Filtering window movement optimisation: In the movement of the filtering window, it can be noted that when the window moves from left to right at the distance of a pixel, the window only has one different column before and after sliding. Namely, the first column in the old window leaves, and the third column in the new window joins in. Similarly, if the window moves down, only one row is different before and after sliding, namely the first row in the old window leaves, and the third row in the new window joins in. So, it is possible to propose an optimisation of the filtering window movement, that is only process the newly added pixels and do not process the already ordered portion in the old window.

Although this optimisation can greatly improve the sorting performance, it has higher requirements for locally ordered arrays. Take the upward and downward movements as an example, the elements are not allowed to move between rows. Since if the row that will be replaced immediately has any element that is swapped to another row, the new row will replace the pixel which should not be left in this move, resulting in making the array of recording pixels in the filtering window disordered. Therefore, in order to satisfy the optimisation conditions of the filtering window movement, in the steps which mentioned in the previous section, although the rows on diagrams are moved, only step A which involving sorting rows themselves exchanges the array elements in the actual operation, the other steps do not exchange any elements between rows.

Suppose that the previous optimisation is based on the method of sorting rows only, the optimisation can be only applied when the filtering window is sliding up and down. Thus, we choose to let the filtering window move up and down for most of the time. When the filtering window reaches the image edge and need to turn, this optimisation is not applicable because the direction has been parallel with the sorted rows. Therefore, whenever the filtering window turns, you need to re-initialise the filtering window array, which means to process this area on a traditional way.

The sorting steps of the improved extremum median filtering method when the filtering window moves down are as follows:

- (a) Remove the upward first row from the old window.
- (b) Sort the new row first, and then add this sorted row to the new window. At this time, the new window is completed, and it is equal to the finishing step (a) of the improved sorting mentioned in the previous section.
- (c) Sort according to step (b) to step (d) of improved sorting mentioned in the previous section.

Take Fig. 3a as a starting point, the diagrams in Fig. 4 show the process from the window sliding down to the new pixel joining in and sorted by rows. In these figures, the numbers underlined are the values in the old filtering window. The values with strikethrough are about to leave the window. The ones with the strikethrough is the new value.

The filtering window moves along the serpentine route, and the specific direction is determined by the width and height of the processed picture. Take the picture whose height is longer than width as an example, in order to reduce turning as much as possible, the window's movement in the image should be: the image begin to move down from the upper left corner. When it reaches the lower boundary, it begins to move right, and then move up from the lower boundary. The distance of each move is a pixel; keep looping like this until the window reaches the right border and then end up with sliding to the right. In order to cooperate with this movement, the sorting in the filtering window should be sorted by rows only. If the picture's width is longer than its height, it first moves to the right and executes it in a similar manner.

Since the values between rows cannot be exchanged, when it initialises the filtering window array each time, the first and the

<u>3</u>	<u>4</u>	<u>5</u>
<u>2</u>	<u>7</u>	<u>8</u>
<u>1</u>	<u>6</u>	<u>9</u>
5	2	7

a

2	5	7
<u>2</u>	<u>7</u>	<u>8</u>
<u>1</u>	<u>6</u>	<u>9</u>

b

Fig. 4 Diagrammatic sketch of improved sorting when the window is sliding

a Before sliding down
b After sliding down and sorting

third columns of the grey value's two-dimensional array obtained by the step (c) in the previous section and the second column (namely the median of each row) of the RGB's two-dimensional array obtained by the step (d) should be separately recorded into a one-dimensional array. It can avoid sorting each column again every time the window moves and further reduce the amount of computation. In fact, in addition to the moments that the algorithm begins implementation and each time the window reaches the image edges and turns, the method of this paper does not need to consider the whole two-dimensional array because there is enough ordered information being recorded in the one-dimensional array. It just needs to take the steps of obtaining the extremum and median of the new row, inserting them into the one-dimensional array and sorting, which make the idea simpler. Compared to the existing methods, this optimising idea needs extra five one-dimensional arrays. However, it does not cost too much because the length of arrays is the window size and the window size of the median filtering is not large, generally 3×3 or 5×5 ; namely the length of a one-dimensional array is 3 or 5.

In conclusion of the two optimisations, the flow chart of sorting in the filtering window is shown in Fig. 5. A detailed description of each step is as follows:

- (a) Initialise filtering window arrays. The extremum median filtering method applied to colour images requires four two-dimensional arrays in total, which are used to store the grey values and RGB values separately.
- (b) Sort every row of each two-dimensional array, and store the first and third column of the grey array and the second column of the RGB array into five one-dimensional arrays, respectively.
- (c) Sort each one-dimensional array, and the extremum of grey values and the rough median of the RGB values will be obtained.
- (d) Determine whether the grey value of the current pixel is equal to the extremum. If yes, it is replaced by the pixel value computed by the median of RGB and go to step (e). Otherwise, directly execute step (e).
- (e) Determine according to the sliding direction and the location of the current filtering window: If it is sliding rightward, it indicates that the window has moved up and down; if it is sliding up and down and does not reach the boundary, it should keep sliding; if it is sliding up and down and already reached the boundary, it should move to the right. Sliding up and down goes to step (f). Otherwise, go to step (g).
- (f) Replace the row in the old window with a new row of pixels (all the four two-dimensional arrays need to be done once). After the new row is sorted, insert the first and third value of the new grey row and the second value of the new RGB row into the corresponding

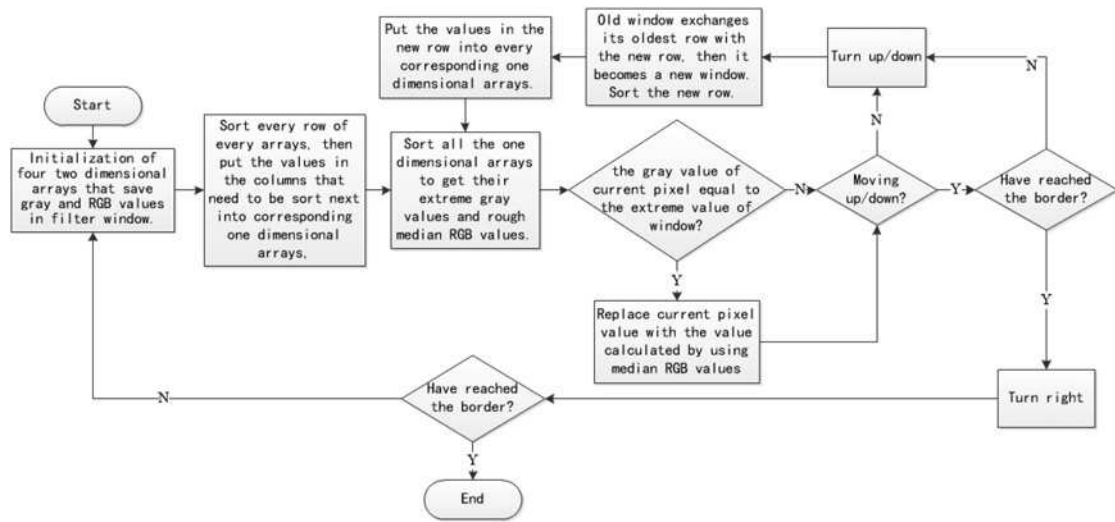


Fig. 5 Flow chart of improved extremum median filtering method

one-dimensional array separately. Sorting each one-dimensional array is equal to just inserting the new value into the ordered array because other positions are already ordered. The extremum of the grey values and the rough median of the RGB values are obtained again, and then go back to step (d).

(g) Determine whether the boundary has been reached. If yes, it means that image has already been traversed and the process is finished. Otherwise, it needs to re-initialise the array and go back to step (d).

Compared with the previous improved sorting method that aims at the interior of the window only, the version of adding the window movement optimisation is equal to completing the step faster, which needs three comparisons to sort the new row as the other two rows are already ordered and do not need to be operated. Thus, it reduces six comparisons. It needs two comparisons for replacing the newly added value into the ordered array and finding the correct position. Thus, during the process of upward and downward movement, it only needs $3 + 2 + 2 + 3 \times (3 + 2) = 25$ comparisons to get the new extremum of grey values in total, which reduces 26 comparisons on the original basis. Compared with the original algorithm, it reduces 95 comparisons. Since, in the window movements, the movements that do not reach the image edges account for the most part. Thus, this optimisation can be applied to all of them and this idea can bring enough benefits in the aspect of time complexity.

In conclusion, the extremum median filtering method after optimisation, compared with the traditional method, greatly improves the efficiency because of significantly reducing comparisons in the sorting process. In the process of acquiring the median, it chooses to roughly obtain the window median for further improving the efficiency. Thus, compared with the existing algorithms, which can acquire the median exactly, there must be some loss in the effects of image denoising. As for whether it is acceptable, the later experiments will prove it.

3.3 Optimised transmission strategy aimed at numerical sensing data

The features of numerical sensing data have been described in Section 3.1.1, and they are not introduced again here.

3.3.1 Existing methods and their limitations: The data transmission strategy proposed in [21] adapts to the characteristics of numerical data, which are roughly linearly changing. It transmits data in the form of 'reference value+change value' and the server recovers data after receiving packets according to reference value and deviation value.

The specific strategy is to divide data into groups at interval of n and suppose data are X . The demarcation points of each set of data are X_0, X_n, X_{2n} and so on, and they are also regarded as reference points. As for the data X_i between X_{kn} and $X_{(k+1)n}$ ($k = 0, 1, \dots, (N/n) - 1$), if $|X_i - X_{kn}| < \Delta$ (Δ is the set threshold), the data at X_i are considered to be substantially stable and give up transmission. X_{kn} will be filled by default when the server does not receive the data at the appointed time so as to improve the transmission efficiency. Instead, if $|X_i - X_{kn}| > \Delta$, the difference-value is sent to the server and the server computes the current value by taking advantage of X_{kn} and difference value. The advantage of doing so is to reduce the sent data volume by sending smaller difference values so as to improve the transmission efficiency. Adjusting the two parameters n and Δ may have an influence on reduction of data volume and loss of data information.

The specific process of this transmission strategy is described as follows:

- Visit the data packet in the current location. If it is a demarcation point packet, go to step (b). Otherwise, go to step (c).
- Transmit the data packet to the server.
- Compute the data difference-value of the current packet and the last demarcation point packet. If the absolute value of the difference-value is greater than the set threshold, go to step (d). Otherwise, go to step (e).
- The difference-value will be transmitted to the server, and the server can calculate the value of the current packet through the difference-value and the previously received demarcation point packet.
- Give up transmitting the packet, and the server can replace the value of the current point with previously received demarcation point packet.
- Determine whether the packet has been transmitted completely. If yes, finish the process. Otherwise, visit the next packet, and go back to step (a).

Although this transmission strategy has a certain rationality and it has also been proved in the related work, considering characteristics of sensor data, it cannot be directly applied to the environment of this paper. There are two main reasons:

- The data of numerical sensors are fragmental. If it is transmitted in the form of packet one by one or time-slot by time-slot. The mobile intelligent terminal must keep sending small packets or keep writing small files to transmit, which will generate great pressure on IO operation of mobile intelligent terminals.

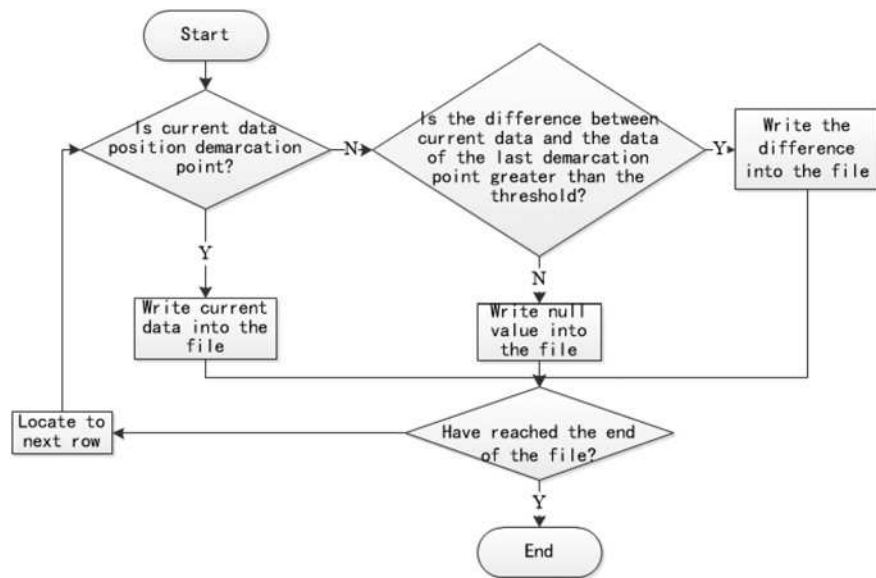


Fig. 6 Flow chart of data compressing process

(ii) The sampling frequency of sensors is not fixed, and the server cannot wait for sensor data coming in a certain moment, which means it cannot determine whether it needs to ignore the data of the time point according to whether the data has been received.

3.3.2 Ideas for improvement: Considering the limitations of the original sensing data transmission strategy, this paper changes the approach. It does not divide big data packets, but performs one more step before writing files, which can achieve simplification of data files and relieve the transmission pressure on the mobile side by reducing data file size and transmission data quantity.

The specific solution of the improved data transmission is that, for each data already stored in the cache, compute the difference-value between it and demarcation-point data in the cache. If the difference-value is greater than the threshold, replace the original data with the difference value. Otherwise, only record the time value indicating that the data does not change much and the other values can be replaced by the demarcation-point data. On the server side, when receiving and reading the csv files, if it reads non-zero data, recovers the data through computing turning point data plus difference value. If it reads the data only including time value, it directly replaces the row with the demarcation-point data. Finally, it stores the restored file to the server side. The purpose of this approach is to optimise the transmission strategy, but in fact it is achieved by letting mobile devices perform one more step of simplification operation. Therefore, it can be regarded as a kind of

data compression. It reduces storage pressure on mobile sides and transmitted data quantity by increasing writing pressure on mobile sides and computation on server sides.

The flow chart of data compression in this transmission strategy is shown in Fig. 6.

A detailed description of each step is as follows:

- Visit the data of the current location. If it is at the demarcation point, go to step (b), otherwise go to step (c).
- Write the data in the current location into the file directly.
- Compute the data difference-value between the current data and the last demarcation-point data. If the absolute value of the difference-value is greater than the set threshold, go to step (d), otherwise go to step (e).
- Write the difference-value into the file. The current data can be obtained later through this difference-value and the recently written demarcation-point data.
- Write the null value of only including sampling time and the current data can be replaced by the recently written demarcation-point data later.
- Determine whether it is the end of the file. If yes, end the process, otherwise visit the next data and go back to step (a).

4 Experiment verification and parameter optimisation

4.1 Experiment environment

The environment information of mobile devices used in this experiment is shown in Table 1.

4.2 Optimisation of the sliding average method

In this experiment, we compare the traditional sliding average method with the improved method. Then, we acquire the threshold n , the number of regional data points, with the best

Table 1 Experiment environment

equipment type	Che2-TL00
CPU	8 core 1.2 GHz
running memory	1.0 GB
storage capacity	8 GB
resolution	720 × 1280
android version	4.4.2
system version	EMUI 3.0

Table 2 Processing time comparison before and after improvement

Dataset size, KB	Processing time before improvement, s	Processing time after improvement, s ($n=1$)	Processing time after improvement, s ($n=2$)	Processing time after improvement, s ($n=3$)
418	9.01	3.93	3.92	4.01
1034	21.34	8.87	8.89	8.85

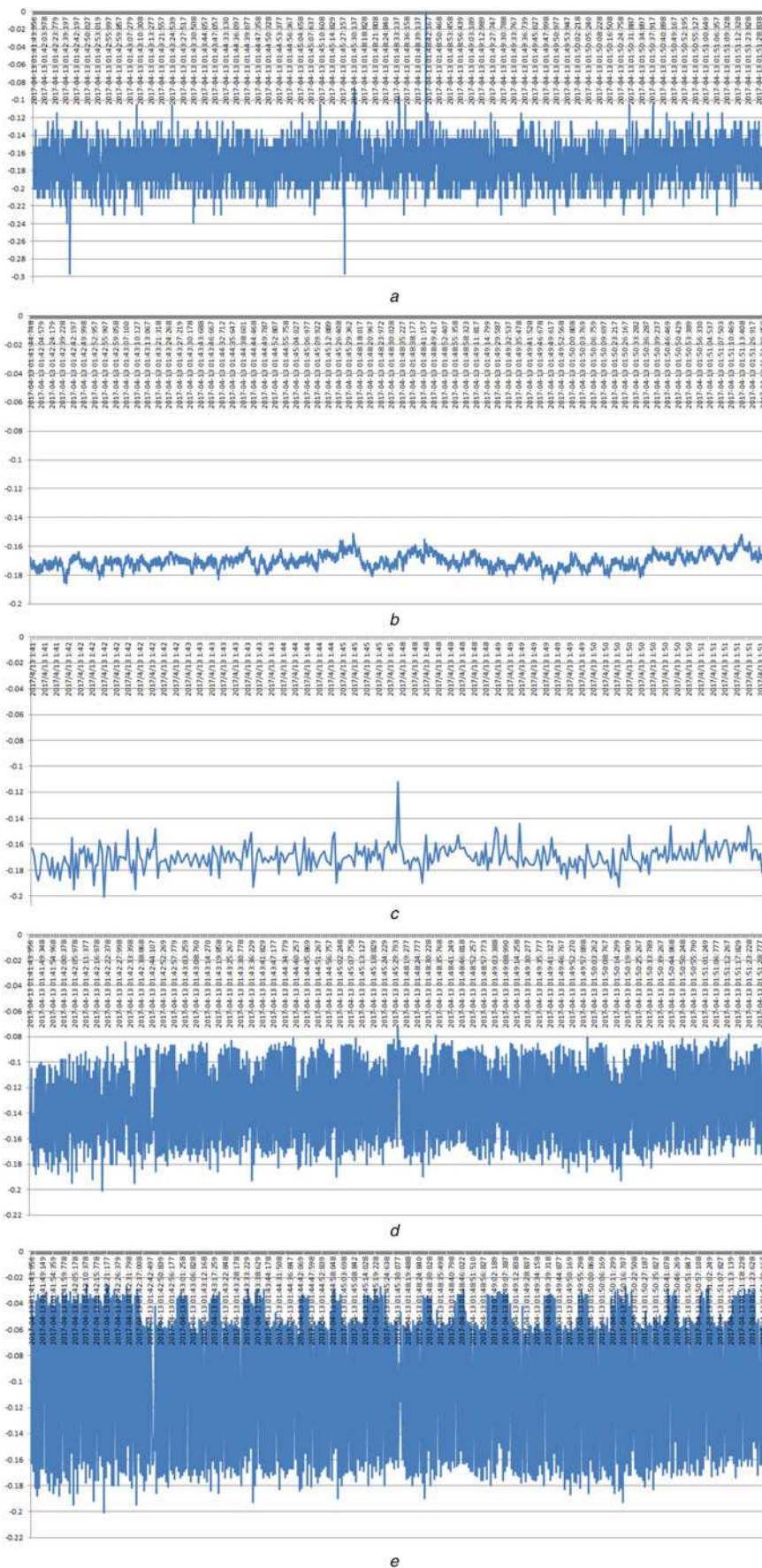


Fig. 7 Comparison between data graphs

- a Data graph before processing (standard deviation=0.019)
- b Data graph after processing by traditional average method (standard deviation =0.005)
- c Data graph after processing by improved method ($n=1$, standard deviation =0.009)
- d Data graph after processing by improved method ($n=2$, standard deviation =0.038)
- e Data graph after processing by improved method ($n=3$, standard deviation =0.054)

smooth result by analysing processing time, processing effects and the size of data files written after processing.

4.2.1 Processing time comparison between the traditional sliding average method and the improved method under different n : Sensors used in the experiment are acceleration sensors. Since acceleration sensors are more sensitive in reading change and their sampling frequency are high, they are able to collect more continuous data in a relatively short time, which is beneficial to compare and observe. The data set is the data respectively obtained in 10 and 50 min using acceleration sensors.

Experiment design: For comparing effects after filtration, this experiment needs to compare the processing time on the same data set. Since the data obtained in two executions must have some differences, it cannot experiment with the data acquired by sensors in real time. We need to collect data using acceleration sensors in the specified time slot. After writing the csv files, we use the algorithm before and after the optimisation to process the data that is read from files.

Experiment results: On different sizes of acceleration sensor datasets, the processing time comparison before and after improvement is shown in Table 2. The n value of the improved algorithm is 1, 2 and 3, which means the maximum number of data points in the same second is 1, 2 and 3, respectively.

As can be seen from the table, if the sliding average method is directly applied to numerical sensor data with intensive data points, it takes longer time. The optimised algorithm greatly reduces the amount of calculation due to only calculating several times for every time interval, and it is significantly better than the traditional sliding average method in the processing time. As for n , the maximum number of data points in time intervals, the effect of this parameter on processing time is negligible.

4.2.2 Comparisons of data-changing trends and standard deviations between traditional sliding average method and the improved under different n : The experiment design and results are shown below.

Experiment design: Draw graphs in the same time slot by using data before processing, data processed by the traditional algorithm and data processed by the improved algorithm. Compare their changing trends and standard deviations to determine whether the optimised sliding average method will damage the information amount of the original data.

Experiment results: Taking the data set collected within 10 min as the standard, we plot the data obtained in the previous experiment of every situations in the graphs in Fig. 7. The abscissa on the graph is sampling time. All data are accurate to milliseconds except the results of the optimised algorithm when $n=1$. The ordinate is the x value of data points, and the standard deviation of x is plotted at the end of each graph title.

Fig. 7a shows that the original sensor data fluctuate very seriously. There are many noise points, and the data are heavily redundant. Thus, it is difficult to know the changing trend of data from the graph. If such data are recorded directly into files, it obviously lacks analytical value and will unnecessarily increase storage and transmission pressure on mobile intelligent terminals.

Fig. 7b shows that the data after processing by the traditional sliding average method. It shows a certain changing trend. The numerical fluctuation is greatly reduced and the noise influences are filtered considerably. Compared with original data, the standard deviation is significantly reduced. It also reflects the enhancement of data stability.

Consider Fig. 7c, the effect of this algorithm when n equals 1 is similar to that of the traditional sliding average method. Data-changing trend is quite close to the data fluctuation interval. The data-changing trend is clearer due to the simplification of data points. As for the standard deviation, it is greatly improved, compared to original data. However, it is worse than the traditional method because of discarding too many data points.

Fig. 7d shows that the effect of this algorithm when n equals 2 is not obvious and the standard deviation becomes larger. The reason is that, when data points within one second are simplified to more than

one data point, the noise points are always calculated along with normal data points. There are little data points in the time slot that is in milliseconds and the weight of each data point is not determined by the outlier detection. Therefore, the smooth effect is poor.

Fig. 7e shows that the effect is even worse when n equals 3. Thus, it is unnecessary to test the larger n and the best n is 1 which means that when the data of every second is simplified to one data point at most, the algorithm works best.

4.2.3 Processed data size comparison between traditional sliding average method and the improved algorithm under best n : The experiment design and results are shown below.

Experiment design: Compare the size of each data file obtained from the experiment in the first part. When $n > 1$, the processing effects are not good. Thus, the part about the improved algorithm in the following table only lists the data size when $n = 1$.

Experiment results and analysis: Data size comparison of the sliding average method before and after improvement is shown in Table 3.

It can be seen clearly that the traditional method lacks control of data quantity due to averaging on each data point. However, compression of data quantity is an urgent need for crowd-sensing applications on mobile terminals. By contrast, when the improved method is recording data, the number of data points per second is controlled up to one which has an excellent effect on data compression.

In conclusion of the above three experiment results, the best n value is 1. In the aspect of processing effect, compared with the sliding average method, the improved method shows data trend more clearly, but with a worse effect on data stability. In addition, this algorithm outperforms the traditional method in the execution time and compression effect. Obviously, because this algorithm has a great advantage of reducing the amount of computation and storage. It is more suitable for preprocessing numerical data on mobile terminals with higher sampling frequency. It can meet the basic preprocessing requirements of rapidly refreshed data and greatly reduce the volume data need to record. Meanwhile, it also can basically reflect data-changing trend and avoid data redundancy.

4.3 Optimisation of extremum median filtering method

In this experiment, we compare effects and performance of the algorithm before and after improvement in two aspects. We analyse processing effects on salt and pepper noise and the processing time with different filtering window sizes.

4.3.1 Comparison of denoising results before and after improvement: The experiment design and results are shown below.

Experiment design: In order to fit the environment of mobile intelligent terminals, the pictures used in this experiment are not the pictures often used in image denoising experiments that are shot by the experiment device and the photo size is 2336×4096 . We add a certain noise ratio of salt and pepper noise to the photo, and then we use the traditional extremum median filtering and optimised algorithm to process it, and finally compare their intuitionistic processing results and PSNR.

Experiment results: The pictures used in experiments are given in Fig. 8. The noise ratio of the relevant noise images is 0.5.

Comparison of PNSR after processing of the two algorithms is given in Table 4.

Table 3 Comparison of Data sizes before and after improvement

Dataset size, KB	Processed data size before improvement, KB	Processed data size after improvement, KB
418	329	16.4
1034	937	36.5

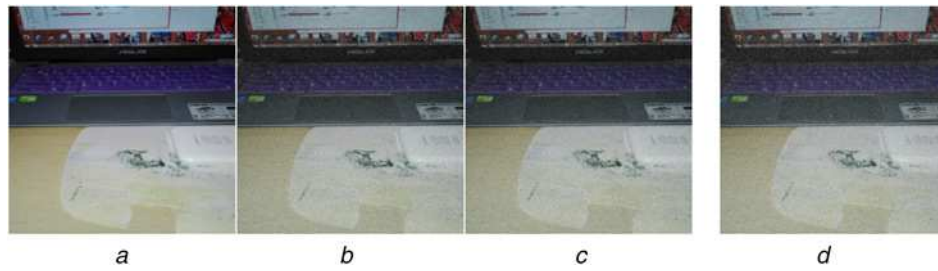


Fig. 8 Comparison of denoising results

a Original picture
b After adding noise
c Original method
d Improved method

Table 4 Comparison of PSNR

Noise ratio	PSNR before improvement	PSNR after improvement
0.3	41.75	40.13
0.5	38.61	37.00
0.8	35.89	34.53

PSNR is one of the commonly used indicators to measure the image denoising effect. In general, the larger value of PSNR, the closer the image is to the original image, and the better effect of denoising is.

Observing and comparing the images and tables, no matter it is from the visual effects, or from the quantitative indicators, the algorithm developed in this paper is not as effective as the original extremum median filtering method due to the rough RGB median used in the sorting. However, the our idea is to sacrifice a little PSNR in exchange for significant performance improvement. Whether the loss in PSNR is acceptable, it needs to be determined by the performance experiment in the next section.

4.3.2 Comparison of processing time with different filtering window sizes: The experiment design and results are shown below.

Experiment design: Use the most commonly used 3×3 and 5×5 filtering windows in the median filtering method. The images with different noise intensities are denoised by using the two algorithms, and compare their processing time.

Experiment results and analysis: The comparison of processing time between the two algorithms with 3×3 window is shown in in Table 5.

The comparison of processing time between the two algorithms with 5×5 window is shown in Table 6.

Comparing the tables and the images, we can see that, since both the algorithms before and after optimisation are the traversal-type

Table 5 Comparison of processing time with 3×3 window

Noise ratio	Processing time before improvement, s	Processing time after improvement, s
0.3	150.73	67.14
0.5	141.20	70.76
0.8	157.05	67.86

Table 6 Comparison of processing time with 5×5 window

Noise ratio	Processing time before improvement, s	Processing time after improvement, s
0.3	382.80	96.76
0.5	381.33	97.83
0.8	372.94	103.79

algorithm, the noise intensity has no effect on processing time. However, the sorting calculation within the window will greatly increase after the filtering window size increased. As the result, the processing time also increases notably. Obviously, the improved algorithm is significantly better than the original algorithm in the aspect of execution speed. In the 3×3 filtering window, the processing time is only about half of the original algorithm. When the filtering window size is increased to 5×5 , the advantage of the improved filtering window movement is more obvious, the processing time is almost a quarter of the original algorithm.

The algorithm in this paper is not as good as the original extremum median filtering method in processing denoising effect. However, the improvement in processing speed is far greater than the disadvantage in processing effect. For mobile intelligent terminals with limited computing power and difficulty in pursuing the best image denoising effect, this algorithm sacrifices a little PSNR in exchange for great performance improvement. The improved algorithm obviously has certain practical value and can be applied to partial related crowd-sensing applications that have a not high requirement for image denoising or need to denoise the photos.

4.4 Optimisation of transmission strategy

This experiment mainly analyses the influence of two parameters on the improved transmission strategy and finds the best parameter value. The two parameters are reference point intervals and the threshold judging whether to record the change value. Sensors used in this experiment are still acceleration sensors.

4.4.1 Comparison of compression effects under different parameters: The experiment design and results are shown below.

Table 7 Comparison of compressed data size with different time segment size

Original data size, KB	Compressed data size when segment size is 5, KB	Compressed data size when segment size is 10, KB
16.4	14.9	15.2
36.5	32.7	32.6

Table 8 Comparison of compressed data size with different threshold

Original data size, KB	Compressed data size when threshold is 0.01, KB	Compressed data size when threshold is 0.015, KB	Compressed data size when threshold is 0.02, KB
16.4	14.9	12.7	11.1
36.5	32.7	27.1	23.8

Experiment design: In this experiment, we use the acceleration data files processed by the optimised sliding average method as data set, and compare data files' size before and after compression. During the experiment, try different combinations of parameters to analyse the influence of each parameter on the compression effect.

This experiment will adjust two parameters. One is, in data compression process, the time interval between each reference point when data are divided into several reference points. The other one is the threshold to determine whether to record current data section as zero value or to record the change value. Namely,

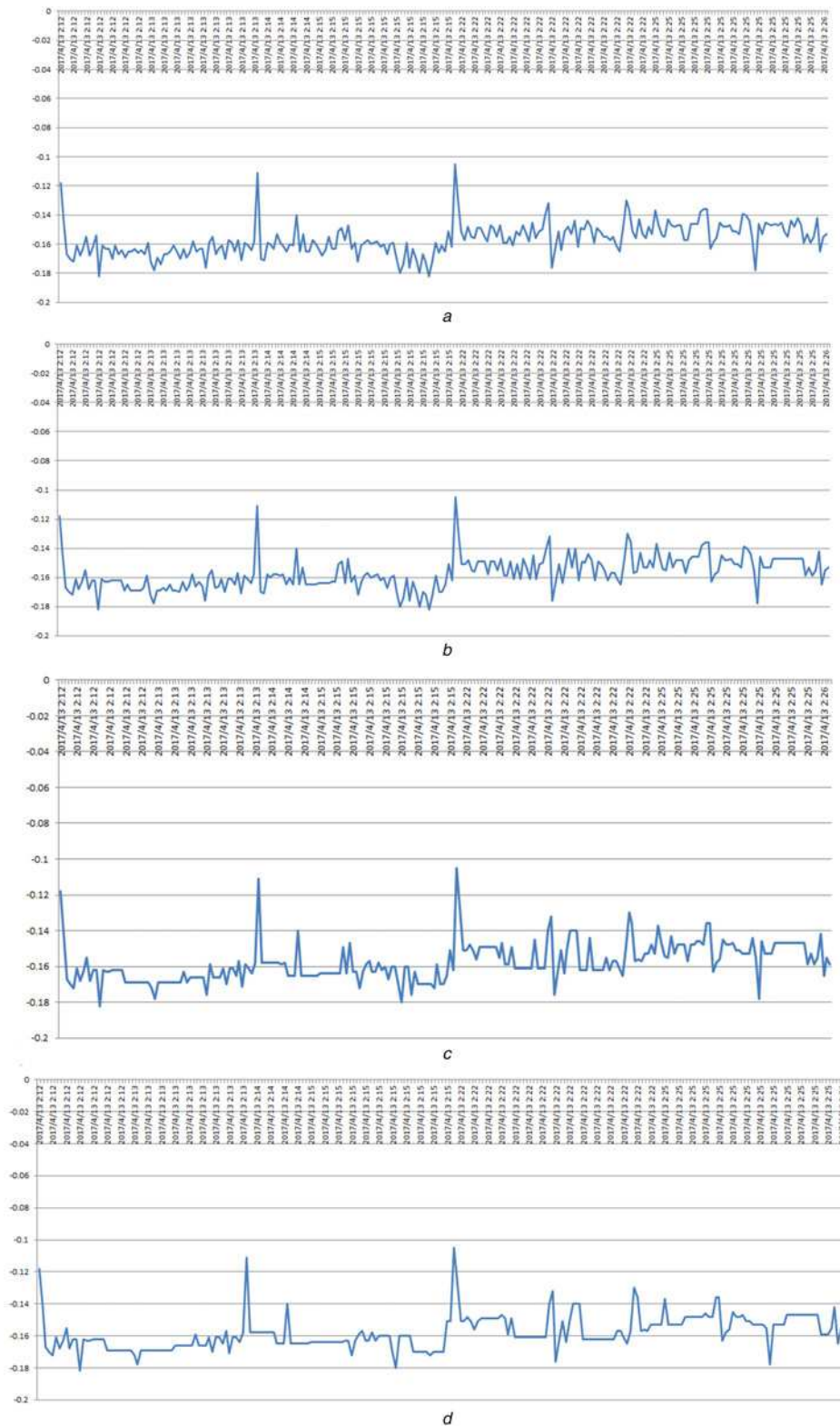


Fig. 9 Comparison between data graphs after recovering

- a Data graph before compressing
- b Data graph after recovering (threshold = 0.01 m/s²)
- c Data graph after recovering (threshold = 0.015 m/s²)
- d Data graph after recovering (threshold = 0.02 m/s²)

when the difference of the data section and the reference point is greater than the threshold, record the difference, otherwise only record the time value.

The data set is the same as that in the previous sliding average method experiment, and is collected in 10 and 50 min, respectively. It has been simplified to acceleration data that is up to one data point in one second after being processed by optimised sliding average method.

Experiment results: When the threshold is 0.01, the effects of compression at different time intervals are shown in Table 7.

The result shows that segment size of reference points has less effect on the compression.

When segment size is 10, the comparison of compression effects on three same files under different thresholds is shown in Table 8.

Observing the above table, it can be seen that the threshold is the real factor affecting the compression ratio. The larger the threshold is, the more data points are recorded as null value, and thereby the file size is reduced. However, a too large threshold may make the file to be compressed too big, resulting in compressed files' distortion. In order to find a threshold that can guarantee both compression ratio and information, the next experiment is conducted.

4.4.2 Threshold optimisation: The experiment design and results are shown below.

Experiment design: Choose the acceleration data collected in 50 min and processed by the improved sliding average method as data set.

Read acceleration data processed by this algorithm and simplified up to one data point in one second, compress it and transmit it to the server side for recovery. Draw the graph according to the recovered data and compare the data distortion before and after compression.

Experiment result: When segment size is 10, after compression and transmission under different thresholds, the comparisons of graphs drawn according to data recovered by the server are given in Fig. 9.

It can be seen from the experiment results that the segment sizes nearly have no influence on compression effects. However, the threshold determining whether to record the change value has a great influence on compression effects. If a too large threshold is used, it will lead to too many ignored data points, resulting in great loss of data information. Therefore, we first choose the threshold according to sensor reading range, and then we use this improved sliding average method. We can finally have an appreciable effect on filtration and compression for numerical sensor data, which greatly reduces pressure on preprocessing and transmission of mobile intelligent terminals.

For acceleration sensors used in this section, the threshold depends on applications' requirements. If there is a high requirement for accuracy after uploading to the server, it is recommended to use 0.015 or even 0.01. If it is acceptable of rough data recovery, choosing 0.02 can get the best compression and transmission efficiency.

5 Conclusions

This paper presented the research and implementation on mobile intelligent terminal data pre-processing methods for crowd-sensing. The existing numerical and image pre-processing algorithms and data transmission strategy were revised and optimised to make them more suitable for pre-processing sensing data on mobile intelligent terminals. The optimisations were based on requirements for crowd-sensing environment and applications. The existing work focused less on processing sensing data with various sampling frequency, and the existing algorithms had lower

efficiency on processing massive sensing data, made sensing data records redundant and had higher overhead on high-pixel image denoising. To solve these problems, we optimised the existing numerical and image data pre-processing algorithms and improved transmission strategy. The experiment results showed that our methods could better pre-process sensing data on intelligent terminals.

6 Acknowledgments

The project was partly sponsored by Guangdong province science and technology planning projects (grant no. 2014A040401018), Guangdong province science and technology planning projects (grant no. 2016B070704010), and Guangdong province science and technology planning projects (grant no. 2016B010124010).

7 References

- [1] Yunhao, L.: 'Participatory sensing', *Commun. CCF*, 2012, **8**, (10), pp. 38–42
- [2] Shuyun, X., Jie, R., Xuesong, Y.: 'The research of smart urban transport system based on PCW', *Electron. Des. Eng.*, 2014, **22**, (20), pp. 49–51
- [3] Shu, L.: 'Research on Traffic Congestion Identification Based on Crowd Sensing', *Wirel. Internet Technol.*, 2016, (4), p. 128
- [4] Lubin, L., Yanmin, Z.: 'Noise collection and presentation system based on crowd-sensing', *Comput. Eng.*, 2015, **41**, (10), pp. 160–164
- [5] Wenle, W., Bin, G., Zhiwen, Y.: 'Crowd-sensing based urban noise map and temporal-spatial feature analysis', *J. Comput.-Aided Des. Comput. Graph.*, 2014, **26**, (4), pp. 638–643
- [6] Wenqian, N., Bin, G., Zhiwen, Y., et al.: 'Campus activity sensing and sharing based on mobile crowd-sensing', *Netinfo Secur.*, 2013, (12), pp. 20–23
- [7] Jiafan, Z., Bin, G., Xinjiang, L., et al.: 'Approach for urban popular event detection using mobile crowdsourced data', *Comput. Sci.*, 2014, **42**, (6A), pp. 5–9
- [8] Yingdang, H., Zhe, L.: 'Data Acquisition and Pre-processing Based on MEMS Accelerometer', *Instrument Technique and Sensor*, 2015, (2), pp. 16–19
- [9] Lianle, J., Yinlong, Z., Bin, X.: 'Data processing techniques on sensors of smart terminals for 3D navigation', *J. Comput. Appl.*, 2015, **35**, (1), pp. 252–256
- [10] Jing, W., Xiaolin, L.: 'Application of RBFNN in outliers filtering of velocity sensor', *Transducer Microsyst. Technol.*, 2014, **33**, (11), pp. 153–155
- [11] Yuanxiang, Q., Liang, D., Kun, Y.: 'Approach for cleaning uncertain data based on information entropy theory', *J. Comput. Appl.*, 2013, **33**, (9), pp. 2490–2492, 2504
- [12] Shi, H., Guanghui, L., Wenwei, L., et al.: 'Outlier detection methods based on neural network in wireless sensor networks', *Comput. Sci.*, 2014, **41**, (11A), pp. 208–211
- [13] Hui, Z., Dongxiang, F., Yagang, W.: 'Ceramic tile image noise reduction on improves adaptive median filter', *Inf. Technol.*, 2015, **12**, (10), pp. 28–34
- [14] Tao, P., Xiaobo, W., Weiwei, Z., et al.: 'Application of improved adaptive median filter algorithm in image denoising', *J. Logistical Eng. Univ.*, 2015, **31**, (5), pp. 92–96
- [15] Ming, Y., Lingling, C.: 'A Linear Prediction Algorithm for Image Denoising', *J. Jilin Inst. Chem. Technol.*, 2014, **31**, (5), pp. 72–79
- [16] Ning, T., Yang, L.: 'Median filtering image denoising method based on noise pixel detection', *Microcomput. ITE Appl.*, 2015, **34**, (5), pp. 35–38
- [17] Liufang, T.: 'The algorithm research of image denoising based on median filter and wavelet transform', PhD thesis, Hebei University, 2014
- [18] Min, N., Zhanjun, W., Yanxiong, N., et al.: 'Fast image median filtering method based on statistical theory', *Electron. Meas. Technol.*, 2015, **38**, (6), pp. 60–63
- [19] Li, M., Cao, Y., Prabhakaran, B.: 'Loss aware sample packetization strategy for improvement of body sensor data analysis'. 2015 IEEE Int. Conf. Computing, Networking and Communications (ICNC), Garden Grove, CA, USA, pp. 673–678
- [20] Jiajun, Q., Jun, Y., Yuanjing, Z.: 'Research on lower overhead reliable transport method in wireless sensor network', *Electron. Meas. Technol.*, 2014, **37**, (1), pp. 92–95, 108
- [21] Mingfu, L.: 'Data filtering and distortion-prevention for power saving in Wireless Sensor Networks'. Communication Problem-Solving (ICCP), 2014 IEEE International Conference, Beijing, China, 2014, pp. 103–106
- [22] Zhongshi, H., Xin, L., Yinong, C.: 'Secondary-diagonal mean transformation partial grey model based on matrix series', *Simul. Model. Pract. Theory*, 2012, **26**, (8), pp. 168–184
- [23] Yixuan, P., Min, G.: 'The Fundamental Principle and Application of Sliding Average Method', *Gun Launch Control J.*, 2001, (1), pp. 21–23
- [24] Min, Y.: 'Digital Image Processing' (China Machine Press, Beijing, 2006)