Developing and Implementing Peak Detection for Real-Time Image Registration

Meng Ma[1], Arjan van Genderen[1] and Peter Beukelman[2]
[1]Computer Engineering, EEMCS, TU Delft, the Netherlands,Mekelweg 4(15th floor),
2628 CD Delft,The Netherlands
Phone: +31 (0)15 2786 217
Fax: +31 (0)15 2784 898
Email: {m.ma|a.j.vangenderen}@ewi.tudelft.nl
[2]Eonic B.V.,Delftechpark 26,2628 XH, Delft, the Netherlands
Phone: +31(0)15 2600 432
Fax: +31(0)15 2600 431
Email: peter.beukelman@eonic.com

Abstract— **Finding known object(s) in static pictures or real-time streaming pictures is always an interesting topic for many applications. A well-known and reliable method is Symmetric Phase Only Matched Filter (SPOMF). After the SPOMF operation, a peak or multiple peaks in the correlated image will indicate the position(s) of the object(s). But the magnitude of the peak(s) can be very different for different images. And it is possible that faulty peaks and mountains have influence on the quality of the detection. In that case, a dynamic floating threshold determination algorithm is necessary to accurately detect the peak(s) in the correlated image. In this paper, a self-adaptive threshold determination algorithm is developed and discussed. An algorithm that maintains a real-time sorting for two sequenced queues in order to save one scan of the correlated image is also developed and implemented. Notice that the threshold can possibly detect some fake peaks that are adjacent to the real one. In order to get the true position of the object, those fake peaks should be removed. An algorithm that removes adjacent peaks is also developed.**

Keywords— **Image Registration, Peak detection, FFT**

I. INTRODUCTION

Symmetric Phase Only Matched Filter (SPOMF) has been proved to be a reliable method for detecting known object(s) in the image. In the previous work, we have developed a simple but efficient algorithm to normalize the spectrum in the SPOMF operation [1]. After transforming the resulting spectrum to spatial domain, there will be a peak or multiple peaks at the location(s) of the target object(s). The regular SPOMF can generate very clean and sharp peak(s) if there is no rotation or scale present. Things get worse when the object is rotated and/or scaled. The magnitude of the peak can be reduced due to the interpolation errors during the compensation of rotation and scale and the noise level of the correlated image will possibly be increased. It is also possible that faulty peak(s) that indicate non-object(s) could arise.

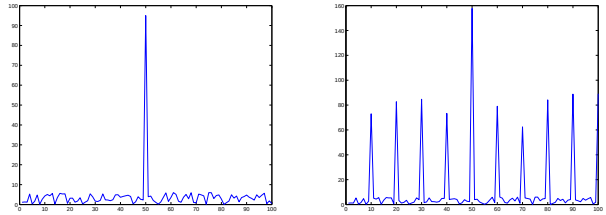
This paper firstly focuses on the design and testing of a self-adaptive threshold determination algorithm. After that, we will discuss an algorithm that maintains a real-time sorting for two sequenced queues in order to save one scan of the correlated image. Note that the threshold can possibly detect some fake peaks that are adjacent to the real one. In order to get the true position of the object, those fake peaks should be removed. An algorithm that does this job will be discussed.

II. THRESHOLD DETERMINATION

The peak detection algorithm should be flexible for different cases. It should be working not only for the regular SPOMF that detects translation without rotation and scale but also for SPOMF that detects the rotation and scale factor to compensate for. Many issues like noisy background or pseudo matches would possibly increase the difficulty of peak detection. Especially in rotation and scale detections, interpolation errors introduced by the mapping from the normal rectangular coordinate system to the log-polar coordinate system will increase the noise level of the correlated image. All those issues make the peak magnitude and the noise level of the correlated image have very different values in different images. Therefore, the peak detection algorithm should be adaptive to the changing situations.

For simulation, we can use 1D signal waves to demonstrate the detection results. Different correlated images can be classified to four basic categories. See figure 1. Here, case 1(a) has a peak with high magnitude in a low noise background, which is the best case for detection. Case 1(b) also has a relatively low noise background but the deviation is high since there are lots of faulty peaks. Case 1(c)

has a high noise level but the level is relatively stable. Case 1(d) has both high noise level and high deviation.



(a)Low noise level, low deviation

(b)Low noise level, high deviation

(c)High noise level, low deviation

(d)High noise level, high deviation

Fig. 1. Four basic classes of correlated image in 1D representation

The noise level is represented by the average value of all the pixels. When the noise level is high, we should also increase the threshold level to avoid too many faulty peaks being detected. The deviation of all the pixels should also have influence to the threshold level. A high deviation should result in a high threshold. Another important term we should take into consideration is the maximum value in the correlated image (usually the true peak). Notice that the actual average of the correlated image is zero, so we should use the absolute value of the pixels to calculate the absolute average. Based on those intuitive ideas, we can write down the threshold determination equation. Let N be the sample number of correlated image and P_i be the value of the ith pixel.

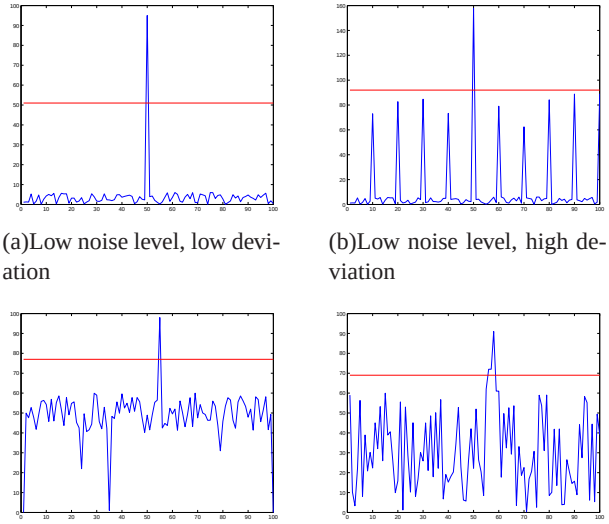
$$Threshold = (Maximum + Absolute\_Average)/2 + K \bullet Deviation$$

$$where:$$

$$Absolute\_Average = \frac{\sum_{i=1}^{N} |P_i|}{N}, Deviation = \frac{\sum_{i=1}^{N} |P_i - Average|}{N}$$

Here the K is the influence factor of the deviation. If the deviation is very low, the threshold will nearly be in the middle of the maximum value and the average value. If the deviation is high, the threshold should also be lifted to avoid faulty peaks exceeding it. If the influence factor K is greater than 1, there exists the possibility that the threshold will exceed the maximum value so that no peak can be detected. For example, if the signal in 1D representation is a continuous increment from 1 to 100 (although not likely to happen in practice), the average is 50.5, the deviation is 25 and the maximum is 100. That makes the threshold become 100.25 that is greater than the maximum value.

From the hardware design point of view, it is good to set the K to a power of 2 so that it only needs a shifter to complete the multiplication task. Figure 2 shows the same signal waves as figure 1 using this threshold determination equation where K is 0.5.



(a)Low noise level, low deviation

(b)Low noise level, high deviation

(c)High noise level, low deviation

(d)High noise level, high deviation

Fig. 2. Threshold determination algorithm (K = 0.5)

As mentioned before, the true average of the real part of the correlated image is zero. That means the deviation calculation equation can be simplified to:

$$Deviation = \frac{\sum_{i=1}^{N} |P_i|}{N}$$

This equation is the same as the calculation equation of the absolute average, so these two can be combined. While both of them are divided by 2 (K=0.5), the combination can remove this division. So the equation could be simplified like this:

$$Threshold = Maximum/2 + Absolute\_Average$$

$$where:$$

$$Absolute\_Average = \frac{\sum_{i=1}^{N} |P_i|}{N}$$

Based on this equation, we can build up dataflow architecture to calculate the threshold. See figure 3. First, the real part of the complex pixels will be fetched into the system. The abs unit calculates the absolute value of those numbers. Then an iterative adder will calculate the sum of

all those numbers by holding the temporary sum in a register that is controlled by the clock signal. An "end_of_data" signal will trigger the register below the adder to release the sum of all pixels to the next stage. In order to calculate the average of those pixels, we need to divide the sum by the number of pixels. The test images we use are all 512 by 512 format, so we only need a right shifter to shift the sum to the right direction by 18 bits. The calculation of the maximum value is relatively simple. The maximum pixel will be kept in a register and it will be compared to each coming new pixel. If the new pixel is larger than the current one, it will be put into the register to replace the old one. It will also be released by the "end_of_data" signal and then shifted by 1 bit to half it. By adding the absolute average and the half of the maximum pixel, we get the threshold.

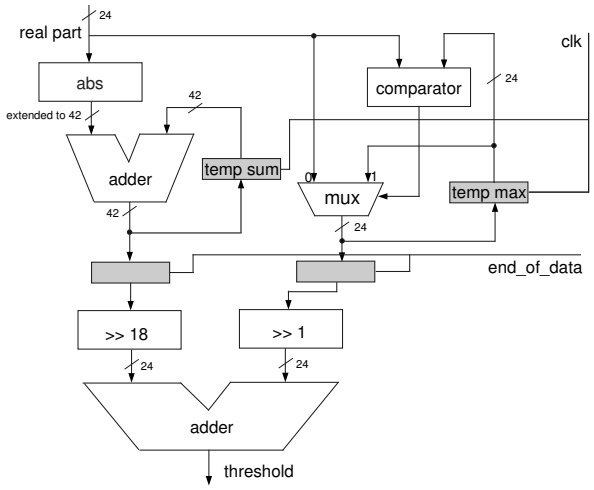


Fig. 3. Dataflow of threshold determination algorithm

III. SEQUENCED QUEUES FOR MAXIMUM PIXELS

A. Sequenced queue

After the threshold has been determined, the peak matrix has to be scanned once again to check which pixels are greater than the threshold and which are not. Those that are greater than the threshold will be recorded as the candidate peaks for further processing. So, totally two scans are necessary here.

This second scan can be saved when the following algorithm is applied. The idea is that we can modify the right part of the dataflow that calculates the maximum value of all the pixels. Instead of finding the maximum, it records the largest 16 pixels. And these 16 pixels are put into a memory queue in order, meaning they are sequenced from largest to smallest. All the information of the pixels is recorded, including their magnitudes, x positions and y positions. This method is safe to apply because normally there are not so many matches (peaks). After the threshold has been determined, the system looks at the smallest element of this queue. If it is smaller than the threshold, which means all the possible peaks are within the queue, the scan for the whole peak matrix is not necessary. If the smallest element is larger than the threshold, then the system will give a "notice" signal to inform the user that there might be peaks missing.

B. Parallel Sorting

A fast search method can be obtained at the cost of some comparators. The method is to compare the new coming pixel to all the elements in the queue concurrently. The compared results are either 0 (if less) or 1 (if greater). Since the original queue itself is sorted, the sequence of compared results is either all 0s or a queue of 1s followed by a queue of 0s, which means the new pixel should be put into the position where the 0 becomes 1. A decoder can translate the resulting sequence to binary representation of the position, which is a 4 bit number. See figure 4.

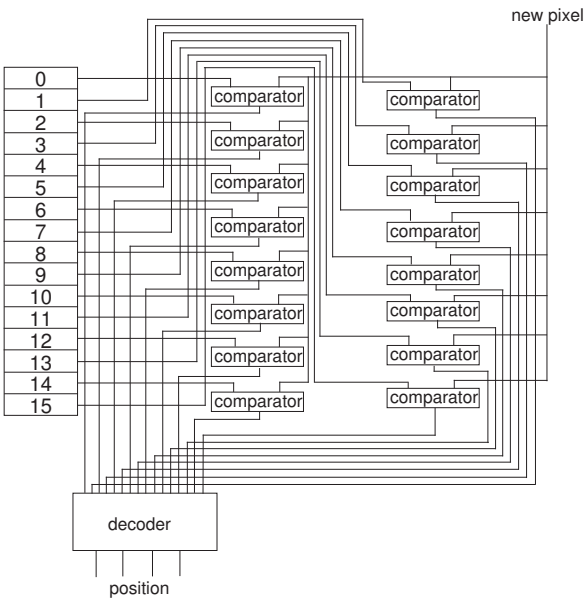


Fig. 4. Comparator network for parallel sorting

The parallel comparisons can be finished in one clock cycle and the updating of the queue requires one cycle. Totally, this parallel sorting unit requires two cycles to finish the job.

C. Hardware implementation of the parallel sorting

The hardware implementation of the queue is shown in figure 5. It needs a 16-comparators network, a 16 bits

shift register, 16 multiplexers and 16 memory locations. The stage signal can be either 0 or 1 and it changes every clock cycle. Note that each pixel is represented by 10 bits for x, 10 bits for y and 24 bits for the peak magnitude. At the first stage (stage = 0), the comparators simultaneously compare the incoming pixel to each element of the memory locations. The compared result called "greater_flag" will be sent to the shift register. The 16 bits of the "greater_flag" are also the write enable signals for the 16 memory locations of the queue. At the second stage, the shift register shifts the "greater_flag" to left by one bit. The "shifted_flag" then will be used as the select signals for the multiplexers. The multiplexers choose either the incoming pixels value or the upper position of the memory locations. In this implementation, no decoder is required to translate the 16 bits flag to a 4 bits position.

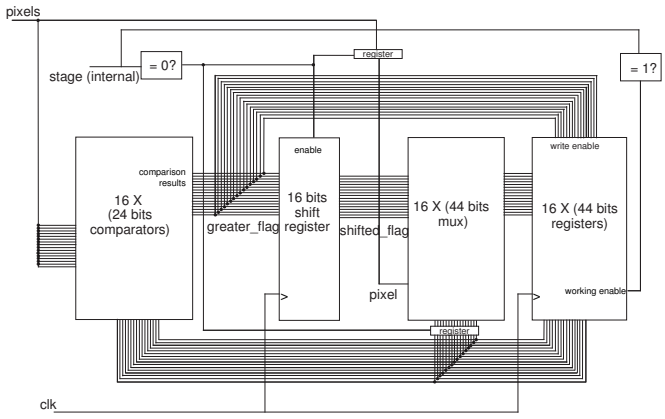


Fig. 5. Hardware implementation of parallel sorting

The position where 0 becomes 1 in the "greater_flag" represents the position where the incoming pixel should be placed. Each bit of the flag is the result of the comparison between the pixel value and the current value at the memory location. For example, a flag of "111111111110000" means that the new pixel is smaller than the first to forth elements of the queue but greater than the fifth to sixteenth. The "greater_flag" will then be used as the write enable signal of memory locations. In this example, the 0 to 3 position of memory will be disabled so that no value can be written to those positions. Old values that are actually greater than the new pixel will be kept. At the second stage, the flag will be shifted left by one bit and used as select signals of multiplexes. In this example, the "shifted_flag" is "1111111111100000", meaning that mux_0 to mux_4 choose the new pixel while the mux_5 to mux_15 choose the values from memory_4 to memory_14 (the content of memory_15 is discarded). Then, the outputs of multiplexers are sent to the memory locations. Since position_0 to position_3 are write disabled, only position_4 are updated to pixels value and position_5 to position_15 are updated to the previous values in position_4 to position_14. See figure 6.

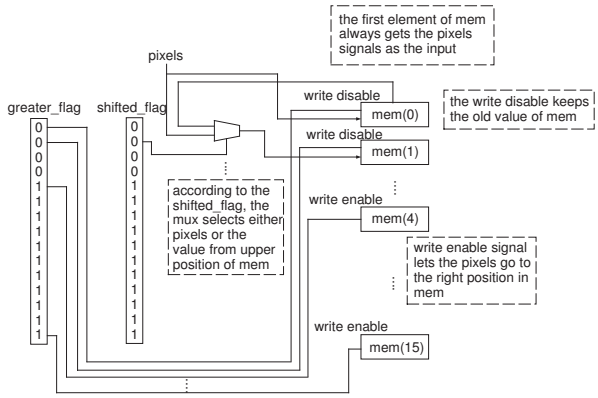


Fig. 6. Mechanism of the parallel sorting

Since one queue requires two clock cycles, two such queues have to be maintained in real time to record 16 maximum numbers respectively. These two queues can work concurrently with the threshold calculation unit. The odd pixels are input to the comparator network associated with the left sequenced queue and the even pixels are input to the right one.

IV. ADJACENT PEAK REMOVING

Sometimes, the peaks that exceed the threshold are not just precisely one pixel position. The nearby (surrounding) pixels also have a high energy (value) that may be higher than the threshold. But these adjacent positions should not be considered as additional translation matches in the search image. The surrounding pixels can be considered as "almost" matches near to the true object position. An algorithm needs to be developed to remove those fake peaks and find the true position (usually the highest one). In order to locate the precise position of the object, the highest pixel among them should be found. (Sometimes the highest one is not the actual position of the object, but it is very close to the real position. Such error is acceptable.

A. Pixel based algorithm

The idea of pixel-based algorithm is to detect adjacent pixels in a way of pixel by pixel. In this solution, not the whole matrix is scanned, but only the surrounding areas of the peaks that exceed the threshold. A square area of 33 by 33 (distance 16) whose center is a detected pixel will be checked to find out if there are other detected pixels within this area. If it finds another pixel, they will be compared to

each other and the pixel with smaller magnitude will be removed. See figure 7. The total number of areas that will be checked by pixel based algorithm is 32 (16 for each queue) while the total number of blocks that the overlapped block algorithm has is $916((32-1)\bullet(32-1))$.

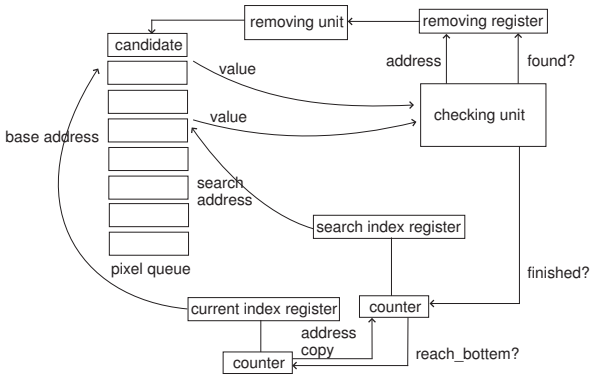


Fig. 7. Schematic of adjacent peak removing unit

B. Chain effect

Although the overlapping block algorithm or pixel based algorithm work quite well in most of the cases, they fail their task in some extreme cases. An example of such a situation in the overlapping block scheme is shown below (figure 8). Here the odd blocks and even blocks are intentionally placed with a slight displacement just to make the figure clearer. Suppose there are pixels in the right parts of blocks 1 to 6, and they are, at the same time, in the left parts of block 2 to 7 (7 is not shown here). Also suppose that the height of those pixels is increasing from block 1 to block 6. Then a problem occurs! The first pixel will be removed compared to the second one within block 2, the second one will be removed since the third one is greater than it within block 3. Finally, all previous 5 pixels will be removed, while only the sixth pixel remains. But the distance between the sixth and first pixel is far more than 16 and they probably indicate different objects. The same case can also happen in the pixel based removing algorithm. Although this kind of situation is hardly possible to happen in practice, at least it deserves some attention.

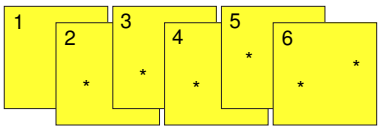
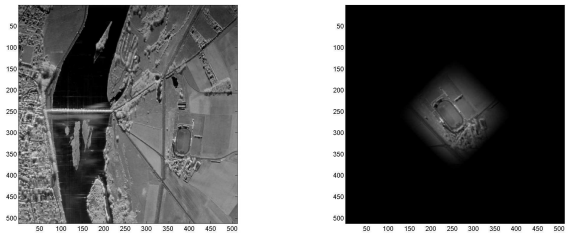


Fig. 8. Chain effect

V. EXPERIMENT RESULT

In this section, a complete detection procedure will be given. The test image is a true satellite photo image. There is a sport center located in the search image and the template image is the rotated and scaled version of this sport center. The template image is not cut from the search image but from another photo. The template image has been Edge-Blurring Circular Filtered in order to remove the cross artifact. The task is to locate the precise position of the sport center in the search image by this filtered template. The search image and template image are shown in figure 9.

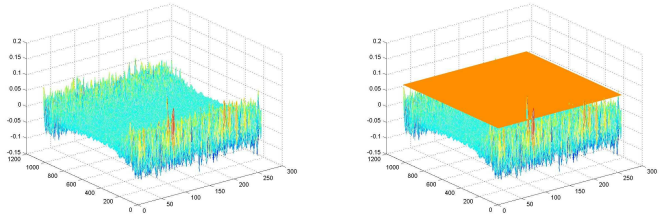


(a) (b)

Fig. 9. Satellite search image and template image

A. Step 1. Detect the rotation and scaling factors

After applying the SPOMF operation to the search image and the template image, we get a correlate image shown in figure 10(a). By calculating the threshold (figure 10(b)) and removing the adjacent peaks, we obtain the rotation angle and scaling factor. The rotation angle is 45 degrees $(\pi/4)$ and the scaling factor is 0.8333 (1:1.2).

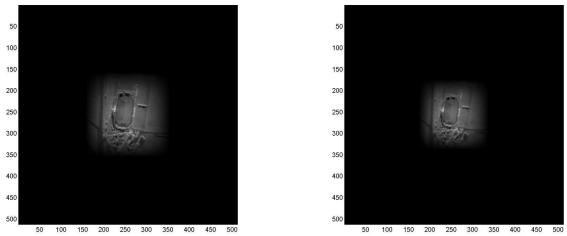


(a) (b)

Fig. 10. Rotation and scale detection

B. Step 2. Rotation and scale compensation

The template image is then rotated and scaled to compensate for the detected the factors. Figure 11(a) is the rotation compensated template image while figure 11(b) is also scale compensated.



(a) (b)

Fig. 11. Rotation and scale compensations

C. Step 3. Detect the actual location of the object

Now, the compensated template image contains the sport center that has the same rotation angle and scale as the sport center in the search image. By applying SPOMF again, we get a clean peak that indicates the location of the sport center. See figure 12.

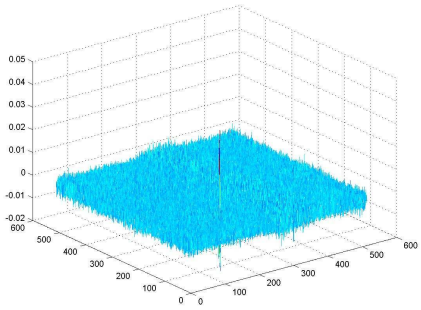


Fig. 12. Detected location of the sport center

VI. CONCLUSION

In this paper, a self-adaptive threshold determination algorithm was developed for use with the SPOMF method. This algorithm can detect peaks in variant environments, independent of the average magnitude of the peak matrix. In order to save another scan of the peak matrix, a dual-queue architecture was designed and implemented. The queues can record the 16 largest of the odd pixels and the 16 largest of the even pixels, in which all the possible peaks are located for most practical cases. Also, an adjacent peak removing algorithm was developed to obtain the location of the detected object more precisely.

REFERENCES

[1] Meng Ma, Arjan van Genderen and Peter Beukelman, *A Sign Bit Only Phase Normalization for Rotation and Scale Invariant Template Matching*, ProRISC 2005

[2] Peter Beukelman and Laurens Bierens, *Real-Time Rotation and Scale Invariant Template Matching in Streaming Images*, GSPx (2004)

[3] B. Srinivasa Reddy and B. N. Chatterji, *An FFT-Based Technique for Translation, Rotaion, and Scale-Invariant Image Registration*, IEEE Transaction on Image Processing, Vol 5, No. 8, page 1266-1271 August 1996.