An Efficient Match-based Duplication Detection Algorithm

Aaron Langille
Laurentian University
alangille@cs.laurentian.ca

Minglun Gong
Laurentian University
mgong@cs.laurentian.ca

Abstract

An efficient algorithm for detecting duplicate regions is proposed in this paper. The basic idea is to segment the input image into blocks and search for blocks with similar intensity patterns using matching techniques. To improve the efficiency, the blocks are sorted based on the concept of k-dimensional tree. The sorting process groups blocks with similar patterns and hence the number of matching operations required for finding the duplicated blocks can be significantly reduced. The matching block detection results are encoded as a color image. This makes it possible to use a set of colour-based morphological operations to remove isolated mismatches, as well as to fill in missing matches. The experiments conducted show the effectiveness of the proposed algorithm.

1 Introduction

As digital cameras and image processing software become more popular and easier to use, digital image forgery is becoming an increasing concern. One of the most common techniques for altering the content of images is by using various copy and paste tools available in most imaging software such as Adobe Photoshop or JASC Paintshop Pro. These tools, including simple copy and paste, clone brush and rubber stamp, create a duplicate region that is identical in size and orientation to the source area. As shown in Figure 1, using these simple techniques, users are able to create perception-altering effects.

An algorithm is proposed in this paper for detecting duplicate regions of the same size and orientation in digital images through block matching. By using Zero-Mean Normalized Cross Correlation (ZNCC) as the similarity measure, our algorithm is robust enough to detect duplicate regions despite image altering compression techniques such as JPEG. A modified k-dimensional tree (kd-tree) construction technique is used to group blocks with similar intensity patterns, thereby avoiding an exhaustive search and improving the efficiency of the algorithm. A set of novel color-based morphology operations is also proposed to refine the detected duplicate regions and to improve result readability.

Figure 1: A cropped section of an image that has been altered in a realistic manner. An object from the scene has been copied and overlaid on another part of the scene.

1.1 Related Research

Two papers in particular [PF04, FSJ03] directly address the issue of detecting duplicate regions in digital images. In [FSJ03] the authors discuss various techniques for determining whether duplicate regions exist and displaying the location in an output image. One technique uses an image segmentation approach where blocks which are sorted lexicographically. This sorting places identical blocks side by side in an array. The sorted blocks can then be easily searched and identical blocks flagged to produce a rough representation of the duplicated regions. They attempt to augment the success of the sorted block approach by searching for blocks that are not only exact matches but reasonably close matches using Discrete Cosine Transforms (DCT). Popescu [PF04] implements a similar technique where blocks of pixel data are compared for similarity. Their algorithm makes use of Principal Component Analysis (PCA) instead of DCT.

Our method similarly segments the input image into blocks. However, different from the above two approaches, we sort different blocks using a kd-tree based technique that effectively groups blocks of not only identical but also similar intensity patterns. After sorting, a common matching technique is used to measure the similarity of neighbouring blocks in the sorted block array. The results of detected duplicate regions are encoded using a color image. Two novel colour-based morphology operations are then applied to the result image so that the accuracy and readability of the detection result can be improved. All of these steps combine to form a simplified and effective algorithm.

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

2 The Matching-based Duplication Detection Algorithm

The algorithm to be discussed detects duplicated regions from a source image based on the following assumptions. The duplicated regions:

- come from the same image.
- are not scaled or rotated.
- are moved from the sources by an integer amount.

Since duplicated regions generated by image editing tools, such as clone brush and rubber stamp, satisfy these constraints, these assumptions do not limit the application of the proposed algorithm.

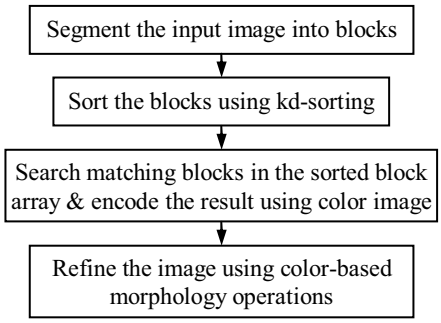


Figure 2: Simplified overview of algorithm flow.

As outlined in Figure 2, the algorithm begins by segmenting the image into relatively small overlapping blocks. For each block, we need to determine if another block of reasonable similarity exists. While an exhaustive search and comparison would provide the most accurate results, the time required to complete such a search would be prohibitive. Instead, our algorithm sorts the blocks based on pixel information using a kd-tree based sorting technique. By sorting we ensure that reasonably similar blocks are located in close proximity. As a result we need only search within a variable-sized neighbourhood instead of the exhaustive search. The matches found are encoded as a color image. After all blocks have been processed a refining operation, consisting of a series of colour-based morphological operations, is applied to remove mismatches in the detection results.

2.1 Creating the Blocks

We begin by segmenting the image into relatively small, overlapping blocks of size $B \times B$. This segmentation process is illustrated in Figure 3. For a given input image with resolution $M \times N$, the number of blocks, N_B, created by this process can be calculated as:

$$N_B = (M - B + 1) \times (N - B + 1)$$

In order to save memory space the pixel intensity data (Figure 3b) is not stored in the block data structure. Instead we store only an array containing the top left pixel of each block (Figure 3c). The intensities of pixels in the block can be referenced from the original image as needed.

The block size is an algorithm variable affecting the accuracy of the algorithm. When larger block sizes are used, mismatches are less likely to occur since more pixels are used for similarity evaluation. However, fine details in the duplicate regions may not be detected when the block sizes are larger than the scale of the fine details. Conversely, using smaller block sizes allows more details in the duplicated regions to appear, but the algorithm will be more sensitive to mismatches due to the smaller area of comparison.

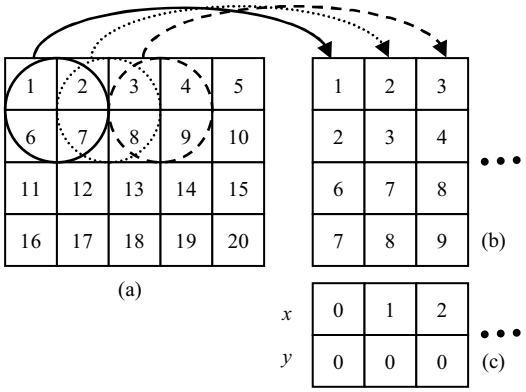


Figure 3: A 5x4 pixel image (a) is segmented into 2x2 pixel blocks. The blocks are logically organized as vectors in array (b). In order to conserve memory space, only the coordinates of the top left corner of each block are actually stored in array (c). Blocks can be referenced when needed using the coordinates and the original image.

2.2 Kd Sorting

A straightforward approach for finding the best match for a given block is to compute the similarity measurement for every pair of blocks produced from the image. The complexity of such an approach is $O(N_B^2)$. To reduce the computational cost we use a kd-tree based sorting approach to group blocks with similar intensity patterns together so that for a given block we need only search within a local neighbourhood, referred to as *Neighbourhood Search Size* (*Nss*). Therefore the resulting algorithm has a complexity of $O(N_B * Nss)$, where $Nss << N_B$.

Kd-tree is a well documented nearest-neighbour sort and search method [FBF77, Moo91, Yia93, TF00]. An outline of the construction of a kd-tree is given in [TF00]. In order to minimize the memory required the actual kd-tree structure is not implemented. Instead our algorithm uses the idea of the kd-tree, together with the quick sorting technique, to create a sorted array of blocks which, as mentioned in section 2.1, is simply an array of top left pixel coordinates. The psedocode of this process is given in Figure 4.

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

As shown in Figure 4, the kdSort procedure recursively splits the block array into two parts each time along different dimensions. Please note that the recursive process terminates when size of the array is smaller or equal to the neighbourhood search size, N_{ss}. This is one way of improving efficiency of the algorithm without affecting the accuracy of the result — since for a given block, the matching step will check all blocks within the neighborhood of size N_{ss}, the order within N_{ss} distance does not affect the result of the algorithm.

```
void kdSort (int d, int start, int end,
     Block data[]) {
  if ( end-start <= Nss )
    return;
  Set p to the split value for dimension d;
  pos = partition(start, end, p, d, data);
  kdSorting((d+1)%NB, start, pos, data);
  kdSorting((d+1)%NB, pos, end, data);
}
int partition(int lower, int upper,
     int pivot, int dim, Block data[]) {
  while (lower <= upper) {
    while (data[lower][dim] <= pivot)
      lower++;
    while (data[upper][dim] > pivot)
      upper--;
    if (lower < upper)
      swap(data[lower], data[upper]);
  }
  return lower;
}
```

Figure 4: Pseudocode for kd-sorting array of blocks.

The actual splitting operation is conducted by the partition function, which is similar to the in-place partition procedure used in quick sort. Since the in-place operation can be performed in linear time, the kd-tree based sorting approach is as efficient as a straightforward lexicographical sort. The average case complexity for kd sorting is $O(N_B \times \log(N_B/N_{ss}))$.

The value that is used to iteratively split the array in two parts is referred to as the "split" value. Talbot [TF00] suggests that the split value can be selected based on the desired node distribution pattern. We tested three different approaches for calculating the split value. The mean and median approaches use the calculated mean and median of the pixel data at the appropriate dimension. A histogram-based approach calculates the histogram distribution of the intensity and then uses the valley of the histogram as the split value. In the end, the mean split was chosen as it provides a satisfactory node distribution at a greater efficiency than median or histogram-based.

When completed, the kd sorted array will place blocks with the most similarity directly adjacent to one another; as the array is traversed in either direction away from any given block, dissimilarity between the starting block and the current block will likely increase.

2.3 Creating the Output Image

By this point the algorithm has logically segmented the input image into vector*ized* blocks of pixel data and sorted the array of blocks. We are now ready to begin searching for matching blocks that have similar patterns. To measure the similarity between two blocks various similarity/dissimilarity measures can be used including sum of square differences (SSD), sum of absolute differences (SAD), and cross correlation (CC). This paper uses the Zero-Normalized Cross Correlation (ZNCC) as described in [JL01]. We found that ZNCC is robust enough to match similar patterns in the presence of minor noise and intensity variations caused by lossy JPEG compression.

The results of the ZNCC measure is a value between -1 and 1, where -1 is completely dissimilar and 1 is identical - the higher the result, the more similar the two blocks are. In our case the vectors to be cross-correlated are formed by the intensities of pixels within the blocks. For color images, a pixel's intensity is computed using the average of the three colour channels (RGB).

```
set znccThreshold;
for each block i in sorted block array B {
  highZncc = 0;
  for (j=i+1; j<i+Nss; j++) {
    Z = calculate zncc(B[i],B[j]);
    if(Z>=znccThreshold && Z>highZncc) {
      highZncc = Z;
      bestX = B[j].x;//Y-coordinate of B[j]
      bestY = B[j].y;//Y-coordinate of B[j]
    }
  }
  if(highZncc!=0) {
    Xoffset = bestX - B[i].x;
    Yoffset = bestY - B[i].y;
    colour1 = encode(Xoffset,Yoffset);
    colour2 = encode(-Xoffset,-Yoffset);
    set pixel (B[i].x, B[i].y) to colour1;
    set pixel (B[j].x, B[j].y) to colour2;
  }
}
```

Figure 5: Pseudo code of the neighbourhood search and output image creation steps.

As described in Figure 5 for each block B_i in the array we compute the cross correlation between B_i and B_j, where $j=i+1, \ldots, i+N_{ss}$. If the result of the cross correlation is less than the specified ZNCC threshold or the previously found maximum ZNCC value, the pair is ignored. This allows us to keep only the best matches within the N_{ss} neighborhood and discard the rest. It would be more efficient to simply stop searching after the first valid match. However, informal testing has shown that accuracy is significantly improved by searching the entire

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

neighbourhood and keeping only the best match that occurs between a block and its related neighbours. The computational cost of the matching process shown in Figure 5 is $O(N_B \times Nss)$.

The x and y coordinates of both blocks that produce the highest ZNCC meeting or exceeding the ZNCC threshold are stored. The coordinate difference between the two blocks is then used to create two 32 bit colors, one for representing each of the blocks. The two pixels in the output image corresponding to the x and y coordinates of the matching blocks are coloured using Δx and Δy as shown in Figure 6. This encoding scheme used can uniquely colour different offsets for images up to 2048×2048 resolution (ie. $-2^{11} < \Delta x, \Delta y < 2^{11}$).

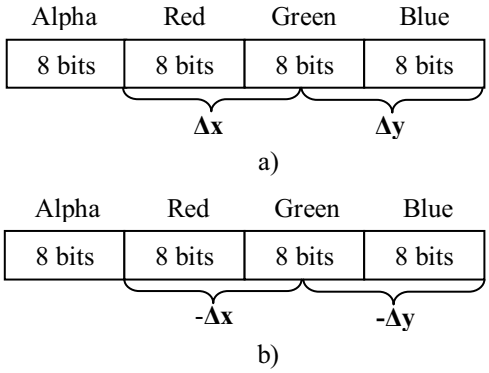


Figure 6: a) Bit shifting is used to encode the calculated offsets, x and y, into 32-bit colour. b) - x and - y are used to create the second colour. The alpha channels are set to 255.

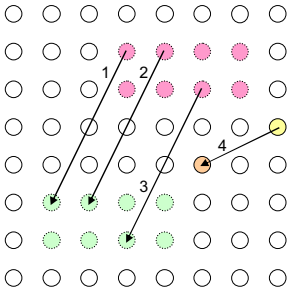


Figure 7: Dotted circles represent forged duplicate region. Since (x1, y1) = (x2, y2) = (x3, y3), two regions of like colour are formed but (x4, y4) is different and forms a lone pair of uniquely coloured pixels (noise). The while color in the encoded color image represents unmatched pixels

By colouring based on offset between the blocks we take advantage of the fact that neighbouring pixels in a duplicate region will have the same offset when correctly matched (See Figure 7). As a result, if a duplicate region is present in an image the copy source and paste destination will appear as two monochromatic clusters of pixels. Coloured noise appears as randomly coloured pixels with few or no neighbours of like colour. Noise can occur for a number of reasons:

- The ZNCC threshold is too low: When the ZNCC threshold is too low random pairs with similar colour that are not actually part of a duplicate region may incorrectly be considered matches. Increasing the ZNCC threshold helps to decrease noise.
- Naturally occurring duplicated regions: Images with large patches of solid colour or naturally occurring patterns can cause noise and even small clusters of like-coloured pixels. Increasing the block size and/or increasing the ZNCC threshold helps to minimize or eliminate these problems.

When all blocks have been processed and all matches have been coloured an output image has been produced. A visual inspection for large groups of like-coloured pixels can be performed to detect the presence of a duplicate region. If too many noisy pixels are present it can make the inspection process more difficult.

Figure 8 shows the output image created by processing the image in Figure 1. While two distinctly coloured areas are apparent, some noisy matches are also present in the output.

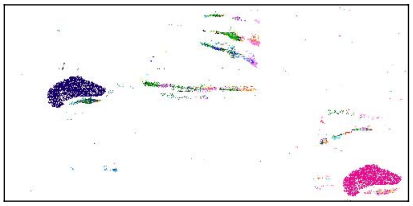


Figure 8: The output image produced from processing the image in Figure 1.

2.4 Colour-based Morphological Refinement

To remove isolated mismatches, as well as to fill in missing matches, an additional morphology-based refinement step is applied after the matching process. Morphology operations are a widely used technique for removing noise in binary images. However, to better remove randomly coloured pixels (noise) and enhance groups of like-coloured pixels, a pair of modified operations consisting of color-based dilation and erosion is used in this paper.

The colour-based dilation using a 3x3 structuring element is defined as follows:

- For any pixel in the image, the immediate 8 surrounding neighbours are gathered and the most frequently occurring non-white colour in the 9 pixels is determined.
- The center pixel is set to the most frequently occurring colour. In the absence of dominant colour or a tie between multiple dominant colours, the center pixel is left unchanged.

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

Our colour-based erosion is implemented as follows:

- For any coloured pixel in the image the immediate 8 neighbours are gathered.
- If any of the 8 neighbours are coloured differently from the center pixel then the center pixel is set to white.

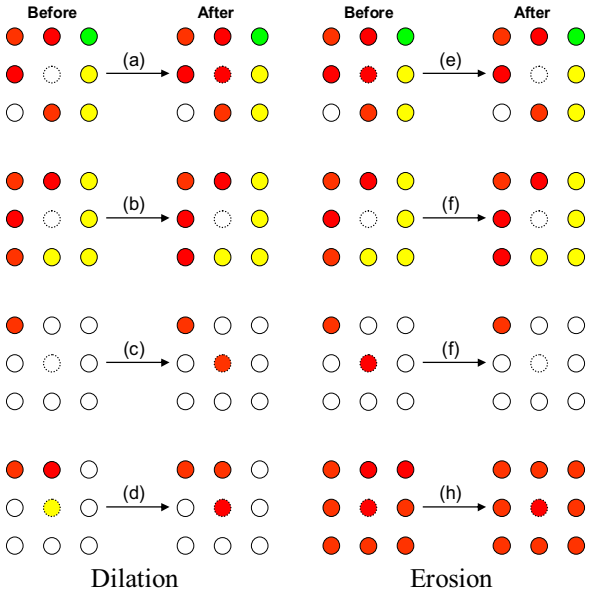


Figure 9: Examples of 3x3 colour-based dilation and erosion. (a) Red is the dominant colour so the center pixel is made red. (b) Since there is no dominant colour the center pixel remains unchanged. (c) For dilation, white pixels are ignored when counting the dominant colour so the single red pixel forces the center to change colours. (d) Even non-white pixels that are not the dominant colour are changed. (e) Multiple colours exist in the structuring element so the center pixel is made white. (f) Multiple colours exist but since the center pixel was already white it is left unchanged. (g) For erosion, white pixels factor into decision making. (h) Other than when it is already white, the only other case where the center pixel is left unchanged is when all of the pixels in the structuring element are the same colour.

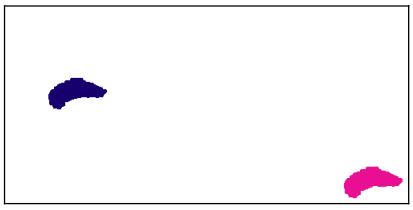


Figure 10: The refined image produced after applying the modified morphological operations to the output image in Figure 8.

In practice, a series of color-based dilation and erosion operations is used to clean up the randomly colored mismatches and fill in missing matches. The sequence that we found to be effective is: dilation, dilation, erosion, erosion, erosion, dilation. As shown in Figure 10, performing this sequence of morphology operations can effectively remove the noisy matches while augmenting local areas of dominant colour.

3 Experimental Results

Several experiments are conducted for evaluating the performance of the proposed algorithm under different parameter settings. As shown in Figure 11(a), an image with a duplicate region is created for testing. In this image, the location and size of the region that was duplicated (the copy operation) are known so that the image can be used as the ground truth for measuring the accuracy of the detection result.

3.1 Test Metrics

The accuracy of the algorithm is evaluated using a binary prediction model similar to the one used in object reorganization research [GHIW96]. In our case, we consider all the matched blocks found by the algorithm as positive predictions and those unmatched blocks as negative predictions. A match found by the algorithm is considered a false positive if it is not inside the actual duplicated region, while an unreported or incorrectly reported match within the duplicated region is considered a false negative.

To distinguish between unreported or incorrectly reported matches within the duplicated region, we further classify false negatives into two groups: unreported false negatives and wrongfully reported false negatives. The first group includes pixels contained in the duplicate regions that are undetected, i.e., the corresponding pixel in the output image has a white color. The second group includes pixels that are wrongfully matched to an incorrect block, i.e., the corresponding pixel in the output image has a colour other than the known correct colours.

As a result, in this paper the False Positive Rate (FPR), Unreported False Negative Rate (FNR_u), and Wrongfully reported False Negative Rate (FNR_w) are defined as:.

$$FPR = \frac{\#\text{ of matches outside of the duplicated region}}{\text{total }\#\text{ of pixels outside of the duplicated region}}$$

$$FNR_u = \frac{\#\text{ of unreported matches in the duplicated region}}{\text{total }\#\text{ of pixels in the duplicated region}}$$

$$FNR_w = \frac{\#\text{ of wrong matches in the duplicated region}}{\text{total }\#\text{ of pixels in the duplicated region}}$$

$$FNR = FNR_u + FNR_w$$

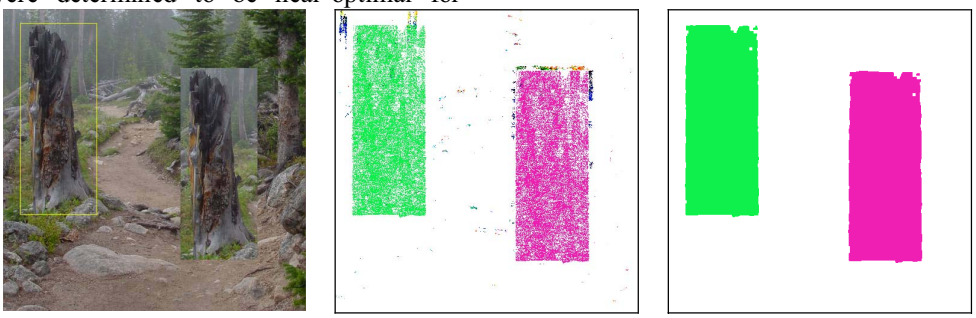
3.2 Effect of the ZNCC Threshold Parameter

The first test evaluates the performance of the algorithm under different ZNCC threshold values. The test image is save in JPEG format (compressed image is

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

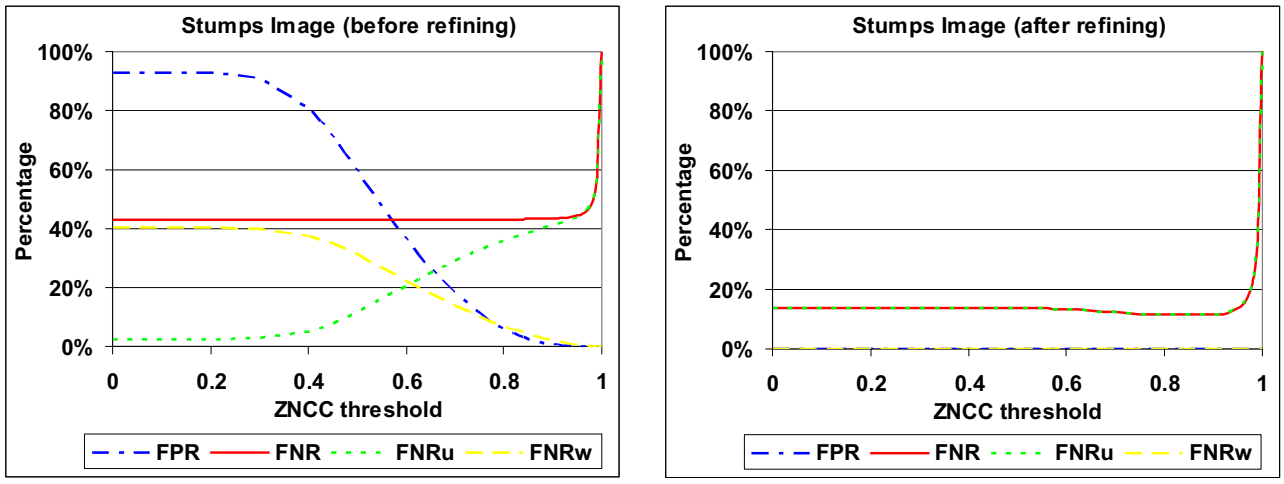
~38.1% of the uncompressed image size).

For this testing, the block size, *B*, was fixed at 10 and the neighbourhood search size was fixed at 100 (both of these values were determined to be near-optimal for accuracy and efficiency in informal testing). ZNCC threshold values varied from 0 <= ZNCC <= 1. The *FPR* and *FNR* of the result under different ZNCC threshold are computed and plotted in Figure 12.



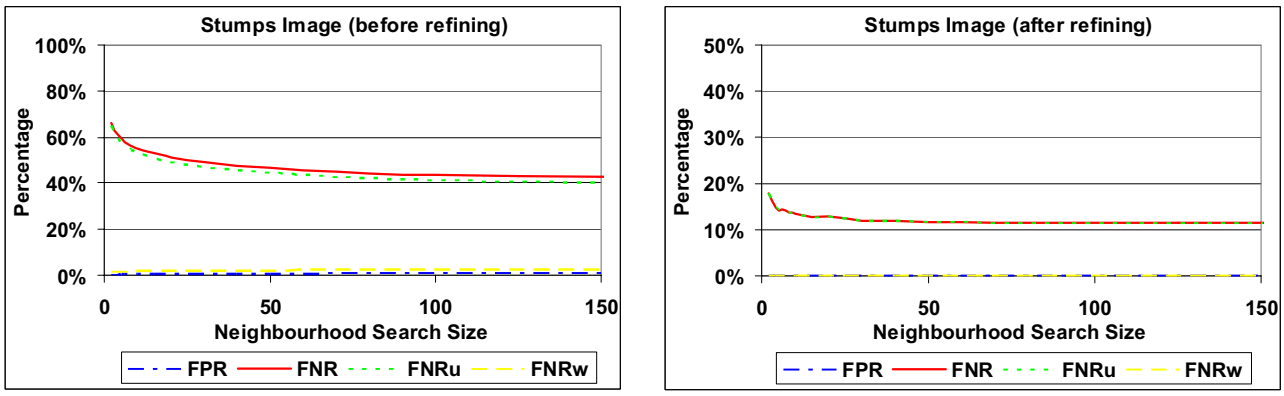
a) b) c)

Figure 11: a) The "stumps" control image; the highlighted rectangle shows the source of the duplicate region. b) the output image created by processing the control image (ZNCC = 0.9, block size = 10x10, neighbourhood search size = 100). c) The refined output image after the modified morphological operators have been applied.



(a) (b)

Figure 12: False positive (FPR) and false negative rates (FNR) at varying ZNCC values before and after image refining operations.



(a) (b)

Figure 13: False positive (FPR) and false negative rates (FNR) at varying neighbourhood search sizes a) before

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

image refining operations and b) after image refining operations.

The following trends can be observed from the graphs in Figure 12:

- In the unrefined output, FPR remains constant at ~92% but begins to decline between ZNCC 0.3 and 0.4 and decreases to <1% by 0.9. This suggests that increasing the ZNCC threshold can effectively reduce the number of false positives detected
- When the ZNCC value is low there are more pixels that are incorrectly matched than are unmatched in the known duplicate region. As ZNCC passes 0.4 the number of incorrectly matched pixels begins to drop since fewer acceptable matches meet the threshold requirement. At the same time, the number of pixels left unmatched begins to increase, also caused by the increased threshold requirement.
- The refining operations provide an effective method for augmenting the results in the final image. The false positive rate (coloured noise outside of the detected duplicate regions), in most cases, tends towards zero.

Please note that the *FNR* and FNR_u curves are overlapped on Figure 12(b) due to the fact that *FNR* is almost equal to FNR_u. This indicates that after refining the majority of errors in the duplicate region are caused by unreported matches. FNR_u does not tend toward zero due to detection issues at the border of the duplicate region.

Similar to *FPR*, FNR_w tends towards zero which indicates that practically all incorrect matches within the duplicate region (coloured noise) have been removed due to the morphological operations.

From these trends we notice that a ZNCC threshold of 0.9 seems to provide the best trade off between *FNR* and *FPR*. Before 0.9 the image is unnecessarily noisy resulting in a high false positive rate. This is problematic for the dilation and erosion operations as areas of like colour will be eroded by nearby noise. After 0.9 the duplicate region becomes increasingly fragmented as matches are overlooked due to the overly restrictive ZNCC threshold.

3.3 Neighbourhood Search Size

For testing the neighbourhood search size we use a fixed ZNCC value of 0.9 and a fixed *B* of 10. Again we calculate *FPR* and *FNR* as described in section 3.1.

We observe the following from Figure 13a:

- FNR continually decreases indicating that more matches are found as the neighbourhood search size increases.
- Without the refinement operations, a good neighborhood search size is around 100. Below 100 we risk missing matches for the sake of improving the running time of the algorithm. After 100, while the number of matches continues to increase, it does so at a rate that is likely not worth the increase in time. Ultimately, if time is not an issue the algorithm should be run with highest possible search size. However, we feel that 100 neighbours represents a good starting point for general use.

We observe the following from Figure 13b:

- In the unrefined results of Figure 13(a) we find that a neighbourhood size of 100 provides a reasonable measure of accuracy and efficiency. However, in Figure 13(b) we note that the dropping rate of *FNR* (which is equal to FNR_u as discussed in the results of Figure 12b) is actually higher and the curve levels off at a smaller neighbourhood size. This implies that when the morphological operators are applied we can use a smaller neighbourhood search size of 50 thus improving the efficiency of the algorithm.

3.4 Qualitative Samples Based on Realistic Forgeries

The following three "credible forgeries" have been created to qualitatively test the algorithm. Figure 14 shows the duplicate regions identified using the following values -ZNCC = 0.9; Nss = 100; B = 10.

4 Conclusion

In this paper we propose an efficient algorithm for detecting duplicate regions. We segment the input image into blocks and search for matching blocks using ZNCC as the similarity measure. To improve the efficiency, the blocks are sorted using a kd-tree based sorting approach. Using the sorted block array, we are able to significantly reduce the neighbourhood search size required to find the correct matching blocks. The matching results are encoded into a color image based on the offset-derived colours between matching blocks. This makes it possible to use colour-based morphological operations to remove isolated mismatches, as well as to fill in missing matches.

Several experiments are conducted for evaluating the performance of the proposed algorithm under different parameter settings. The results show that accurate detection results can be obtained through searching within at most 100 neighbouring blocks in the sorted block array. The novel color-based morphological operations proposed can effectively reduce both false negative rates and false positive rates.

References

[FBF77]Freidman, J.H., Bentley, J.L. and Finkel, R.A., "An Algorithm for Finding Best Matches in Logarithmic Expected Time", in *ACM Transactions on Mathematical Software*, pp. 209-226, 1977.

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE

[FSJ03] Fridrich, J., Soukal, D. and Lukàš, J., "Detection of Copy-Move Forgery in Digital Images", in *Proceedings of Digital Forensic Research Workshop*, 2003.

[GHI96] Gutta, S., Huang, J., Imam, I. F., Wechsler, H., "Face and Hand Gesture Recognition Using Hybrid Classifiers", in *International Conference on Automatic Face and Gesture Recognition*, 1996.

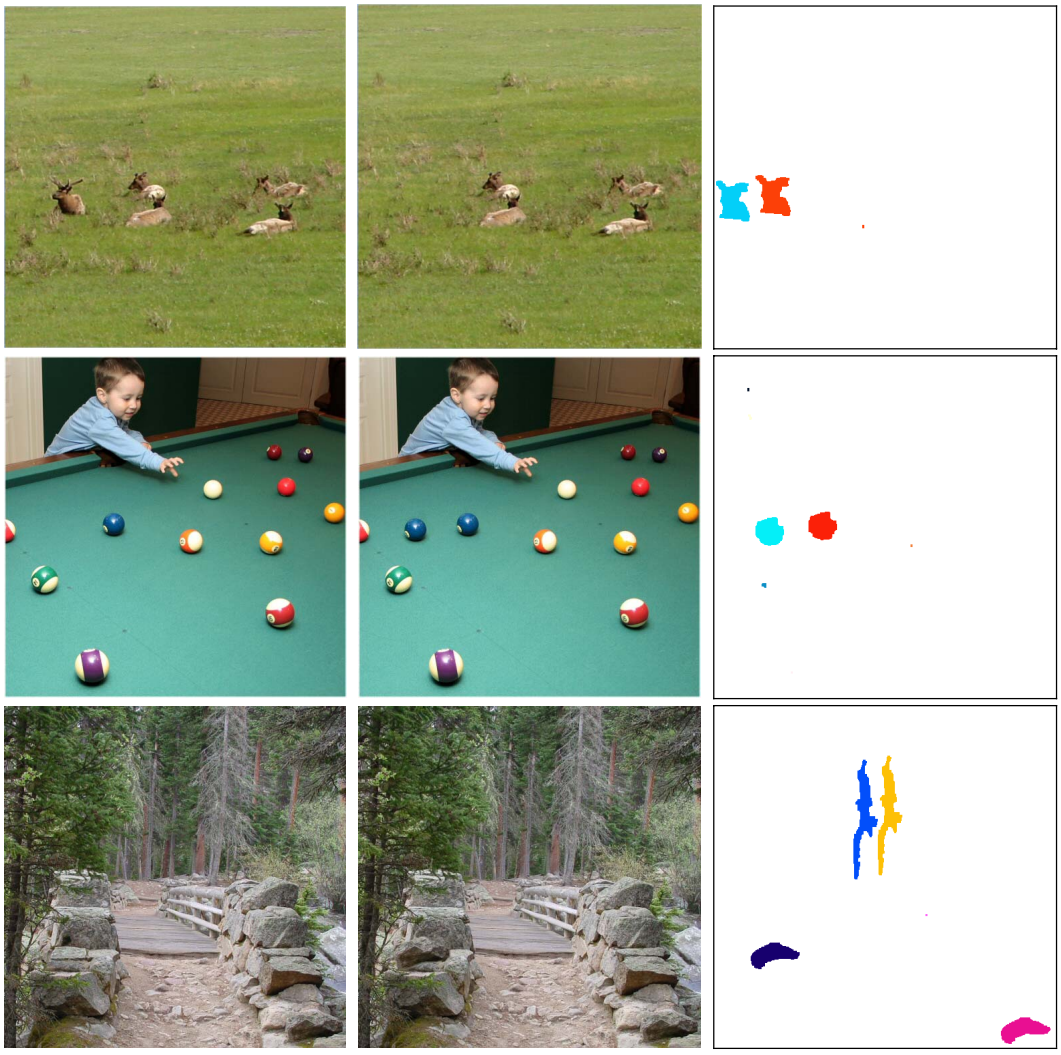
[JL01] Jung, I-K. and Lacroix, S., "A Robust Interest Point Matching Algorithm", in *International Conference on Computer Vision*. 2001.

[Moo91] Moore, A.W., "An introductory tutorial on Kd-trees", PhD Thesis: Efficient Memory-based Learning for Robot Control, Technical Report no. 209, University of Cambridge, 1991.

[PF04] Popescu, A.C. and Farid, H., "Exposing Digital Forgeries by Detecting DuplicateRegions." Technical Report, TR2004-515, Dartmouth College, Computer Science, 2005.

[TF00] Talbert, D. A. and Fisher, D., "An empirical analysis of techniques for constructing and searching k-dimensional trees", in *ACM SIGKD international conference on Knowledge discovery and data mining*, pp 26-33, 2000.

[Yia93] Yianilos, P.N., "Data structures and algorithms for nearest neighbor search in general metric spaces", in *ACM-SIAM Symposium on Discrete algorithms*, pp. 311-321, 1993.



(a) Original Image (b) Forged Image (c) Analyzed Result

Figure 14: Credible Forgeries Created for Qualitative Analysis of the Algorithm.

Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)
0-7695-2542-3/06 $20.00 © 2006 IEEE