On Improving the Performance of the Wallpaper Symmetry Group Classification Algorithm

CMU-RI-TR-05-49

Sravana Reddy [1]

Mentor: *Yanxi Liu* [2]

October 2005

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

[1] sravana@cs.brandeis.edu. Supported by the Computing Research Association's Distributed Mentor Project, 2005.
[2] yanxi@cs.cmu.edu

ABSTRACT

The wallpaper group classification algorithm published in the paper "A Computational Model for Periodic Pattern Perception Based on Frieze and Wallpaper Groups" (Y. Liu, R.T. Collins and Y. Tsin, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 26, no. 3, pp. 354-371, March 2004) is the first computer algorithm for automatic frieze and wallpaper group classification for patterns in real images. In this report, we describe our effort on identifying the limitations of this wallpaper symmetry group classification algorithm. We document the improvements incorporated in the algorithm and code. This report also covers preliminary work done towards ideas for future research to further explore the space of symmetry or near-symmetry 2D patterns.

This work is completed during the CRA Distributed Mentor Project summer research internship, June-August 2005, at the Robotics Institute, Carnegie Mellon University. Sravana Reddy is an undergraduate student from the computer science department of Brandeis University and the mentor Professor Yanxi Liu is an associate research professor of the Robotics Institute in Carnegie Mellon University.

Contents

List of Figures

1. Introduction

Regular lattice structure and pattern symmetry have been inspiring theoretical models in mathematics and physics for a long time. In the late nineteenth century, it was proven that all periodic patterns in n-dimensional space can be classified as one of the n-dimensional crystallographic groups based on the underlying lattices and particular symmetries of the patterns. For patterns repeating along two linearly independent directions, there are seventeen crystallographic groups (known as wallpaper groups), and for patterns repeating along a single direction, there are seven crystallographic groups (known as Frieze groups). [1]

The research done by my mentor in this field centers around computational detection of wallpaper and Frieze group patterns in images [2], [3], [4]. Besides being an interesting computer vision problem in itself, it has several implications in image indexing, gait pattern understanding, human cognition of patterns, and data compression.

The algorithm presented in [2] for wallpaper group detection in 2-d images is the first computer algorithm implemented for frieze and wallpaper group classification on real images. The computational process of the algorithm can be briefly summed up as follows:

1. A parallelogram lattice is found by matching peak points extracted from autocorrelation of the input image pattern.

2. Then a median tile – the smallest generating unit of the pattern, found by taking the median pixel values of each tile – is found.

3. The tile is then rotated and reflected about the necessary eight transformations and compared with the original image to detect which rigid transformations are symmetric. This yields the wallpaper group of the image's pattern.

However, the details of each of the above steps are sensitive to image noise, color information, and irregularities. As a result, the actual computational implementation requires each stage to be robust enough in dealing with a variety of images, which may not have the ideal color distribution for their pattern.

Human vision tends to compensate for pattern irregularities rather well if the image is almost regular. At the level of both the eye (data-driven cognition) and the brain (constructivist cognition), lighting irregularities and image distortions are weighted against the overall perception of the general regularity. [5] However, the methods used in digital image processing differ in some basic ways from human vision.

One of the main restrictions is that the techniques used, at least in the algorithms at hand, are linear, sequential, and to a large extent, independent of the input images. Thus, parameters at each stage need to be finely adjusted to tune out irregularities as much as possible.

Since there is more than one type of non-ideal image at each stage, this involves either building in error-handling approximations or error-contingent function branching. In improving the existing algorithm and code, we pursue both options.

The main stages focused upon for improvement are the lattice and the symmetry detection algorithms.

2. Improving the Lattice Detection

The program by Liu et al., [2] finds potential lattice of a given image pattern using an autocorrelation method. It is based on a novel idea of *region of dominance* where the region (distance) to the next higher peak from each peak is considered more important than the absolute autocorrelation peak values in finding potential lattice points. The algorithm takes progressively larger sets of peaks in the autocorrelation map from the center outwards in increments of 10 and finds the set whose valid lattice best matches the autocorrelation peak values.

Since there is no way of picking the increment non-arbitrarily, moving in steps of 10 runs the risk of skipping some peaks that are significant in the lattice – or, conversely, taking stray peaks that lie outside the lattice intersection points.

Further, the algorithm can be time expensive when a large number of peaks are detected. As the authors point out, "finding the maximal distance among all pairs of nearest neighboring peaks at each iteration… is costly in time."

Analyzing the time complexity of the algorithm:

p = number of peaks

Number of nearest neighboring pairs of peaks $\approx 2p$

Therefore, at best $O(p \log p)$ to find the maximal distance.

If we take iterations in steps of 10, the number of iterations = $p/10$

> For each iteration: we have a complexity of $O(n^2)$ to find the displacement vectors and best lattice in a set of n peaks.
>
> Therefore, over $p/10$ iterations, it takes $O(p^3)$ time to find the best lattice.

We present an alternate approach that is both faster, and more robust in dealing with displaced or missing peaks in the height (autocorrelation) map. The new algorithm also incorporates an automatic verifier of the computed lattice, an important tool for any lattice detection procedure.

2.1. A New Algorithm:

Step 1: Finding the dominant peaks

Find the dominant peaks in the height map and sort them by the region of dominance, as done in the original program.
Set the initial dividing factor (henceforth called f) to be 10.

Step 2: Finding a lattice from a subset of peaks

1. Begin by taking the set of the most dominant p/f points in the size p peak list. For each point, find the nearest (by distance) two points in the list.

2. Store the pair of vectors between the peak and these points.

3. Find the two vectors that occur the most over the whole set of peaks. This gives the vector-pair (*t1* and *t2*) of a possible lattice.

Step 3: Verifying the lattice

1. 'Cut' out the bounding rectangle of the parallelogram defined by the vector-pair found above, starting from the center of the image. Let's call this the generating tile.

2. Translate the outline of the rectangle found above along *t1*, - *t1*, *t2* and - *t2* from the center.

3. Compare the generating tile with the parts of the image bounded by the respective four rectangles found above. If the displacement vector-pairs *t1* and *t2* are correct, the generating tile will be almost identical to each of the four rectangles. If all four correlation scores are high, we assume the displacement vectors are correct, and move on to the next stage. If not, divide f by 2. If $f<2$, quit[3], otherwise go back to Step 2.

The algorithm hence makes, at most, 4 iterations, progressively near-doubling the peak-set. The reasoning behind this algorithm is similar to Liu et al: find the best lattice overlay on a peak set, iteratively increasing the size of the peak set. However, instead of beginning with 8 or 10 peaks and working our way up in multiples of 10, we begin with a tenth of the peak set and exponentially increase the size of the set which, for large sets, results in a smaller number of iterations.

2.2. Complexity

[3] It is inefficient to consider more than the most dominant half of the peaks.

In terms of the number of peaks *p* found by Step 1 (which is the same as the original algorithms'): Step 2 takes $O((p/i)^2)$ to find the two nearest neighbor peaks, then find the two vector pairs that occur the most number of times over the set of displacement vectors. Step 3 takes constant time with respect to the size of the peak set. If we iterate over four peak sets, this is still only $O(p^2)$

It is immediately clear that for small peak sets, Lin et al's algorithm is as good or preferable to the one proposed – it increases the peak set size more gradually and, hence, has a better chance of finding the correct lattice with an optimal set while our proposed algorithm might end up having to deal with a lot of redundant peaks at the stage when it finds the correct lattice. However, for a large peak set, that problem frequently occurs when the displacement vectors are small relative to the image size. When the intensity range is small, our algorithm is a lot more time efficient. It is also better equipped to handle missing peaks in an otherwise near-regular distribution.

2.3. Implementation

In light of the above, the best middle ground would include both algorithms and use either one depending on the number of peaks found (the old one for a small number, and the new one for a larger peak set). We pick a threshold number of peaks – around 150 (we noticed that this is the size after which the algorithm really starts slowing down).
For peak sets of sizes less than 150, we use the old algorithm, and then check the lattice generated using the verifier outlined in Step 3 above.

If the verification is not satisfactory or the peak set size is larger than 150 to begin with, we proceed with the new algorithm.

2.4. Results

Of the 190 images, the old program detected 42 incorrect lattices. The results from the new algorithm are shown below:

Image Category	**Drawings**	**Photographs**
Total Images	75	115
Wrong lattices from the old program	5	37
Wrong lattices from the new program	4	29

Correct on previous failures	2	19
Wrong on previously correct detections	1	15

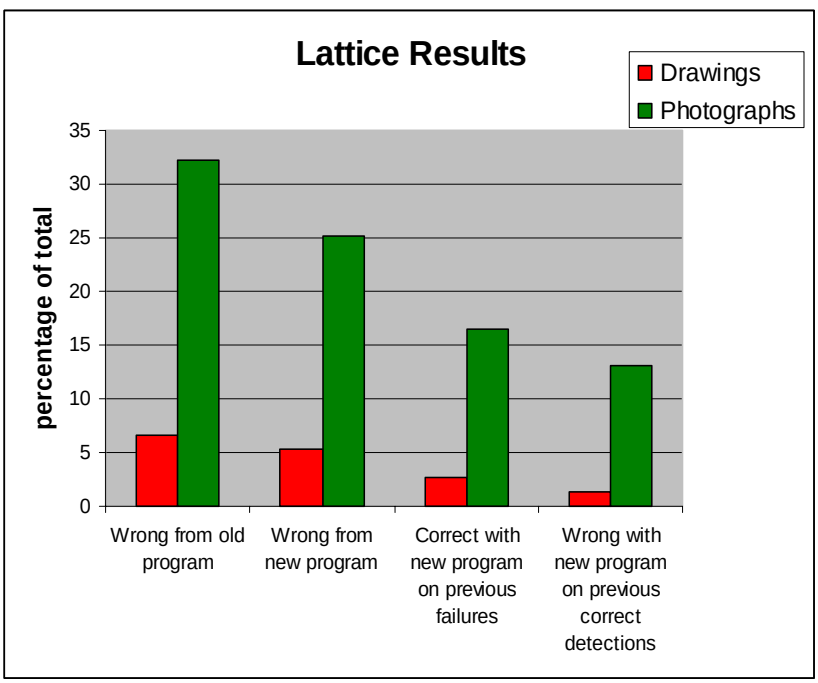


Figure 2.1: Comparison of lattice detection failures of old and new algorithms

2.5. Limitations of the new algorithm

From the results recorded in the table above. The "correct on previous failures" (19) and "wrong on previously correct detections" (15) seem to suggest that the new method may complement the old instead of being strictly better. The **three** main reasons this algorithm still fails on certain images are:

1. The self-verification is done using SSD comparison between the central tile and its four lateral neighbors. However, this is not a perfect indicator of regularity. A pattern could be visually regular, but still contain certain deviations that result in an imperfect match between the tiles. Deviations could be in intensity (particularly in photographic images where the lighting is not uniform), in slight irregularity among tiles, or in spatial orientation; again, a problem in photography where there are perspective distortions. Thus, a correct lattice might fail to pass the verification test, resulting in smaller and less accurate lattices. The image shown in Figure 2.2 is one such instance.

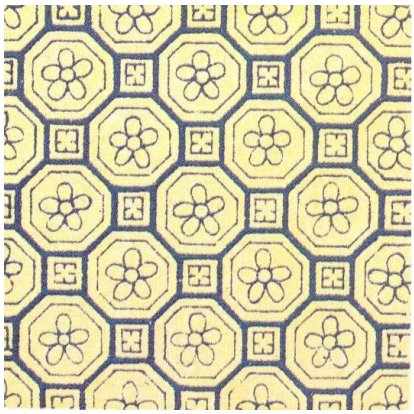
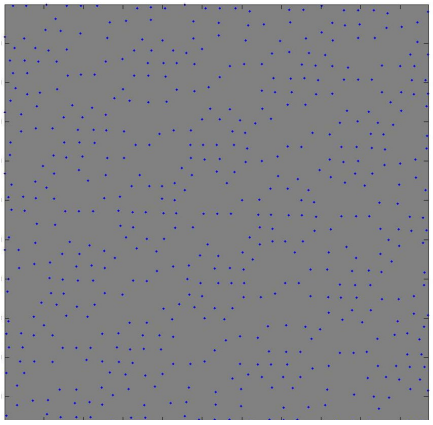


Figure 2.2a: The image above has a very regular pattern tiling with few or no irregularities

Figure 2.2b: Thus, a satisfactory autocorrelation map is found, with potential to yield a correct lattice



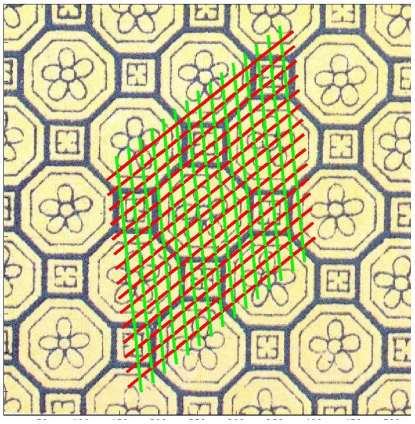
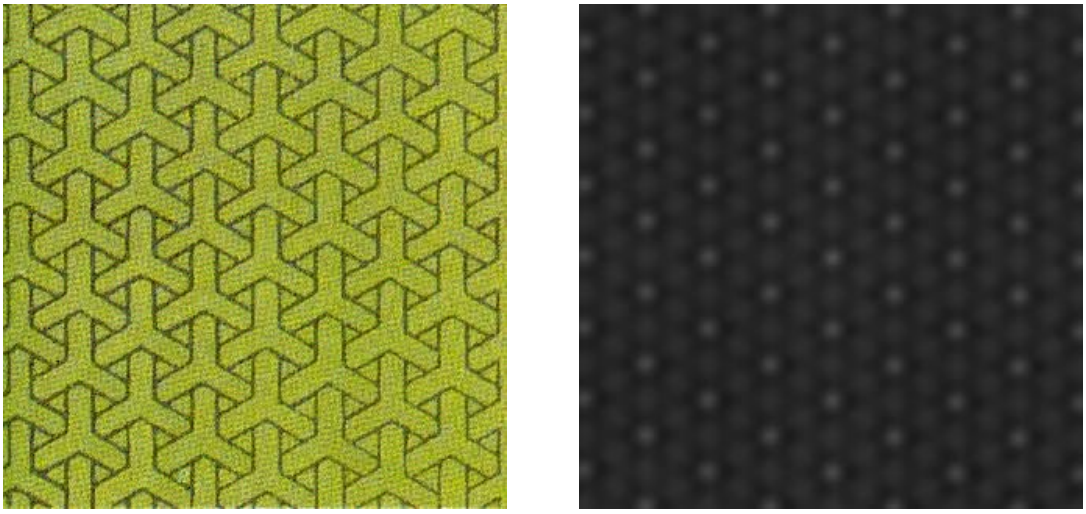


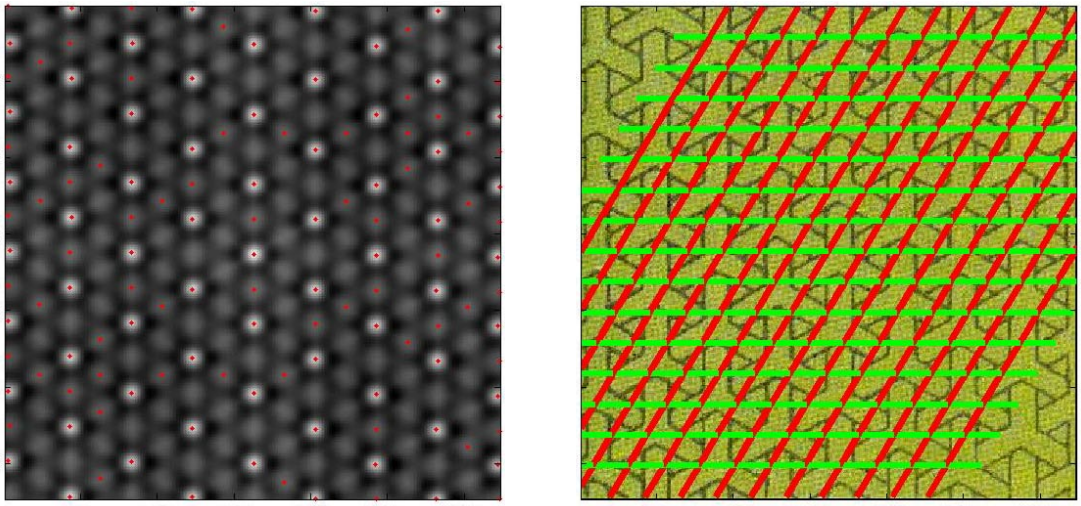
Figure 2.2c and d: However, tile translation along the correctly detected displacement vectors does not yield an exact match, due to subtleties in matches. The program believes the pattern is not regular over that lattice, and iteratively increases the peak set size, trying to find a 'better' match, resulting in more peaks than necessary (left) which then results in an incorrect lattice (right).

Yet, it is clear that some sort of self-verification should be implemented to improve the dynamic efficiency of the program. Future work might improve upon this by either deciding on a threshold value that will more accurately indicate correctness or, better still, finding a time-efficient way to compare every tile with every other and averaging the match scores.

2. For the sake of speed, we double the peak set size on every iteration. This runs the risk of skipping sets that are the best candidates for yielding the correct lattice. A better compromise between speed and efficiency – perhaps by improving the vector detection on an insufficient peak set – would be desirable.



Figures 2.3a and b: The autocorrelation map (right) on the image (left) is regular and a good set of peaks that generate the expected lattice can be found



Figures 2.3c and d: However, after trying fewer peaks than required for the lattice, the program takes too *many* peaks (left), which yield shorter displacement vectors (right) than are correct

3. The algorithm presented was aimed to correct a specific type of lattice failure, i.e. one where the autocorrelation has the potential to yield a good lattice, but fails to do so. However, there are reasons of failure that require a completely different approach – images where the autocorrelation itself is unsatisfactory owing to intensity or pattern distortions; take the edge-image discussed in Section 3.1 as an example. This requires a method that does not depend so much on autocorrelation.

Figure 2.4a: A pattern with a large amount of color irregularity

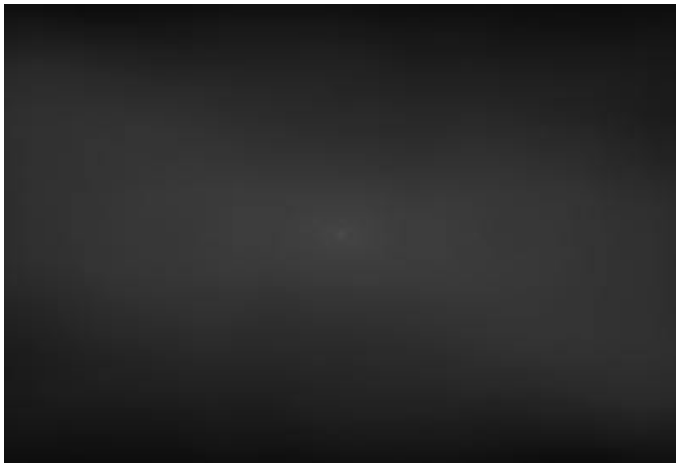


Figure 2.4b: Such irregularities cause a highly inaccurate (or rather, undesirable) autocorrelation map

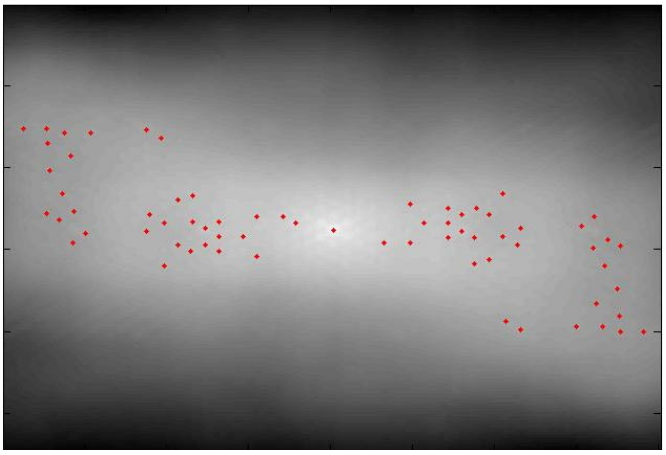


Figure 2.4c: It is, therefore, inevitable that a bad peak map will be found, owing to the autocorrelation failure…

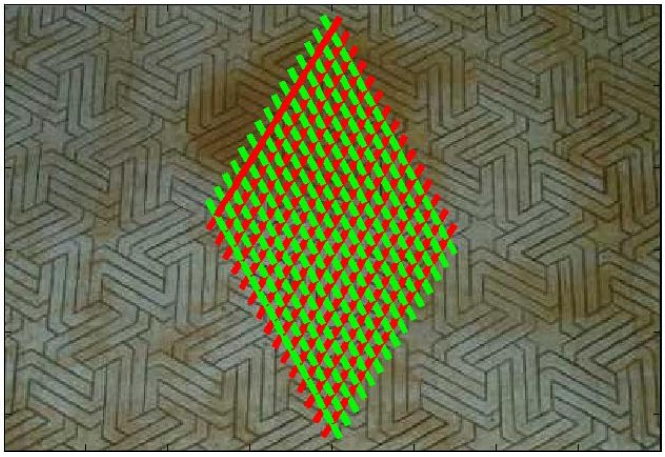


Figure 2.4d: … which naturally results in a poor lattice. The peak map does not correspond in any way to the expected lattice and, thus, it is futile to try to find the displacement vectors using this map as a generator.

3. Improving the Symmetry Detection

The symmetry detection procedure assumes a chi-square noise distribution. The tile, as defined by the lattice, is rotated or reflected through each of the eight transformations and superimposed on the image. A symmetry on a rigid transformation exists when there is a good registration between the rotated/reflected tile and some part of the original image.

The program calculates relative correlations using a sum of squared difference (SSD) measure on the grayscale image. However, this is inefficient for certain kinds of images without the assumed noise and intensity distributions:

a. Those with large areas of homogeneity, so that only a small fraction of the tile (the edges) characterizes the symmetry

b. Those with near-symmetry, i.e., a very small, but visually distinguishable area of the tile is significant in negating a possible symmetry

c. Those with color ranges of ambiguous intensity, so converting into grayscale for comparison causes loss of color – and hence symmetry – information.

a. and b. have similar problems: SSD weighs all pixels in the image equally. A large fraction of corresponding pixels will be interpreted as a good symmetry about the given rigid transformation, even if there are areas in the image that clearly preclude a symmetry. The problem with c. is that standard RGB to grayscale converters average the R, G and B values to get the intensity. This may result in intensity values that match closely even when the colors at those points are vastly different.

3.1. Edge Weighted Symmetry Detection

We begin by identifying images where the regularity of a pattern is primarily characterized by the edges. This involves performing a Sobel edge detection on the median tile and calculating the fraction of the edge pixels to the total area (henceforth called the 'Edge Fraction'). A small fraction indicates what we shall term a 'Rare Edge Image,' one where the edge pixels weigh more than the rest in characterizing the pattern.

Examining the Edge Fraction of several images, a threshold value is decided, below which symmetry detection is performed only on the edge image.

There are two possible ways of comparing the edge image of the tile under rotations/reflections with the complete edge image. One technique is to use a simple SSD comparison. This is problematic because edge images are extremely sensitive to slight registration offsets. For one thing, the edge detection algorithm itself may not yield an edge image most compatible with the perceived symmetry. Further, even accurate edge marking might not cause the features to overlap perfectly in case of similarity.

A better method, therefore, is the Chamfer edge registration/comparison method. This allows for errors in both edge detection and registration. Correct groups were detected for seven of the edge images using the Chamfer method. Unfortunately, it has a complexity of, at best, $O(n^2)$ in terms of the number of pixels n. Faster approximations to the algorithm are possible, including hierarchical and segmented variations; but it is hard to

score the metric appropriately. Thus, the simple Chamfer algorithm was implemented as well as the SSD, which takes over in case of very large images.

3.2. Other Metrics

Other measures of similarity were also implemented along with SSD, including correlation. However, these did not give very different results than SSD since they are essentially the same principle (pixel by pixel comparison).

3.3. Score Interpretation and Clustering

The symmetry detection methods give numerical score-values that should indicate the relative extents of similarity. The next step is to identify which of these scores indicate a good similarity and which do not. This is not a trivial problem, since the score range can vary widely depending on the type of image.

The old code used a simple k-means (k=2) clustering to separate the scores. While this is the most efficient single algorithm for the purpose, it does fail on certain score distributions. (Of course, an ideal similarity measurement method would give clearly indicative scores, but that is an almost impossible target.)

To improve the clustering, certain parameters were added. For instance, it was noticed that a lot of distributions that were clustered around 1 did not separate well under k-means – a threshold of 1 is now used to cluster these distributions.

A few learning algorithms were also applied to the distributions, but none gave much better results than the k-means.

3.4. Nearest Possible Group Detection

To compensate for the clustering problem, a little margin was added to the code that finds the group from the score-values. In most of the failures, the problem was with just one score being misclassified as indicative of a similarity (‘good’) or not (‘bad’). The code was extended to drop the worst score classified as ‘good,’ or take into consideration the best ‘bad’ score, then consider the group that these sets would yield. The results of this are output along with the rest of the scores and group predictions.

3.5. Results

Since we did not greatly alter the basic algorithm as much as add flexibility to it, no previously correct images were detected wrong after the changes. The results of the new procedures implemented are summarized below:

	Total Images	Wrong on group detection with old code	Number improved by clustering	Number improved by edge detection	Number of correct nearest possible groups
Drawings	75	29	5	4	5
Real Images	115	41	10	7	4

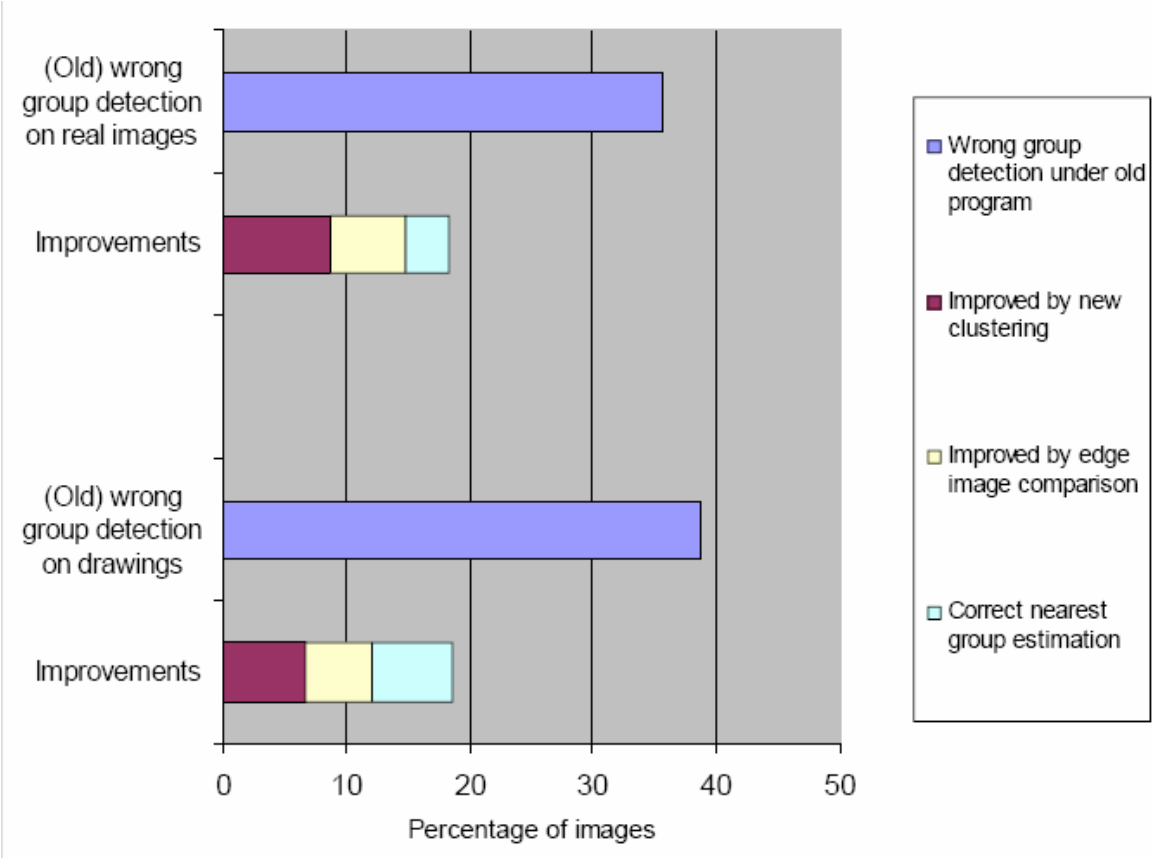


Figure 3.1: Results of modifying symmetry group detection segments

4. Code Readability and Technical Corrections

Various changes were made over the course of the project to improve the readability, speed and organization of the code as well as the output.

An unnecessary nested loop for bulk pattern analysis was optimized into a single loop, thus reducing the running time by *1/the number of images*. The scope of the input image formats was extended to jpg as well as bmp files. The output of the program is now generated as a set of well-organized html files with labels for the image analysis pictures and a clear indication of the group detection results.

The condition for P1 (no symmetry) was also found to be too liberal – a lot of score sets which indicated a symmetry separation were classified as P1 because of the lax variance margin. The condition for P1 is now set to checking if all the scores lie below 1 or above 3, which does not give any false positives.

Finally, a synopsis of the program's individual functions and their organizational structure was documented into a "readme" file for future programmers working with the code.

5. Wallpaper Group Detection in Patterns under Global Distortions

An important follow-up to the work on symmetry group detection is the paper on patterns under affine transformations. [3] This is important in terms of extending the pattern-classification method in question (i.e. based on group theory) to real world textures that are not necessarily photographed from an orthogonal perspective. The paper discusses the theory of 'skewed symmetry groups' for regular textures under affine transformations.

A further extension of this concept is to consider regular textures under a curvilinear transformation and research the possible group-theoretic classification schemes on them.

The first step to this is to find an approximate mesh of displacement vectors throughout the image. This could be done by using the techniques employed by Liu, Lin and Hays (2004), or by a repeated local autocorrelation peak finding algorithm and vector stitching. If it is known what class of transformations the image has undergone (spherical, polynomial, etc), the curves defined by the lattice points against a parallelogram set can be parameterized. This would yield the actual function defining the transformation and, hence, an inverse mapping back to a parallelogram lattice.

While more than one mapping from the points on the transformed image to a set of points defining a lattice is certainly possible, it's likely that there is a finite set of optimal regular lattices that the curved mesh can be mapped onto. Once the lattice has been 'straightened,' we can perform symmetry group detection on the new straightened image.

Unfortunately, time did not permit an in-depth exploration of this idea. Some preliminary code was written within the program to identify and map curves from sets of points. It might, nevertheless, be interesting to explore this further.

6. Wallpaper Group Distribution of Drawings and Real World Images

Aside from the actual program, we also examined the distribution of images among the 17 groups. (The images are patterns taken from the Carnegie Mellon Near-Regular Texture Database[4].) Do patterns tend to be clustered within a few groups or is there an

[4] http://graphics.cs.cmu.edu/data/texturedb/gallery/

impartial distribution? How does the distribution vary between photographs and images? What are the psychological (or in some cases, physical) reasons for the given distributions and bias? Are some groups inherently preferred among the popular existing patterns, and for what reasons?

The figure below show the number of drawings hand-classified under each of the groups:

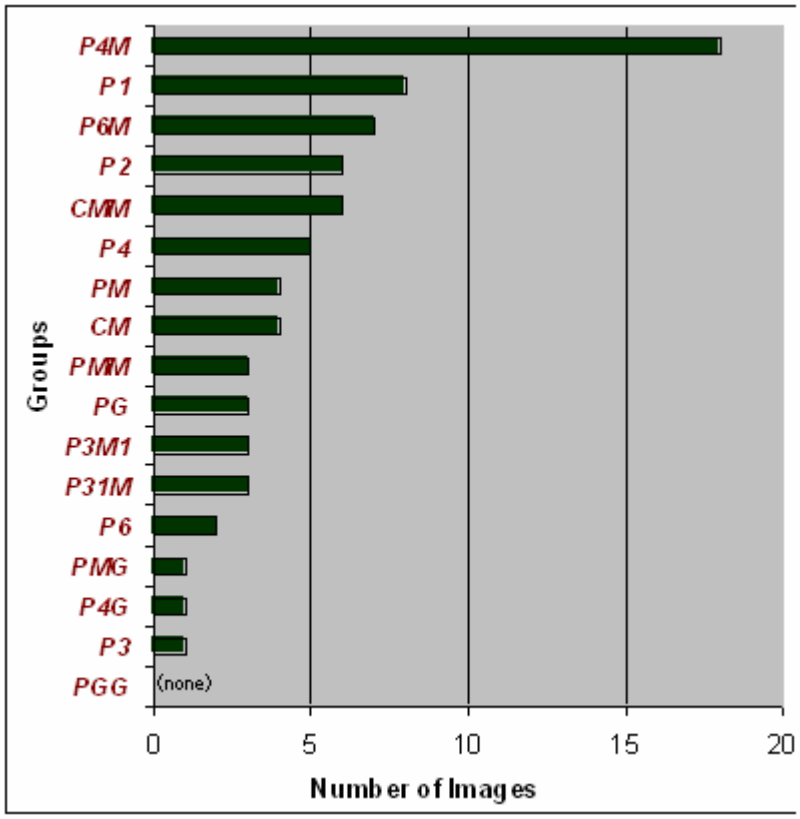


Figure 6.1: Distribution of the ***75 drawings*** in the database

The fraction of P4M patterns is interesting. Since we are working with a relatively small data set, it is not possible to draw accurate conclusions about the occurrence of certain groups in patterned drawings in general.

However, there are some conjectures that can be made. P4M patterns have a sort of 'square' symmetry – they reflect across their edges and diagonals, and have 90^0 and 180^0 rotation symmetries. So it could be reasoned, for instance, that this kind of symmetry allows a viewing relatively independent of orientation, since the patterns stay invariant under four different viewpoints. A more important reason, however, could be the inherent neurological bias towards bilateral and higher order (four- or six-fold) symmetries. This also explains the distribution towards P2, P6M and CMM groups.

Since these images are made by hand, the same kinds of arguments that have been made about the functions of symmetry in art [11] can be extended to account for the pattern-artist's design preference for certain symmetric combinations.

Consideration must also be given to the viewer's preference. For the most part, no special attempt was made to have a uniform distribution of groups when collecting images for the database. Perhaps viewers who seek out symmetric patterns tend to look towards P4M-like patterns (since those symmetries are the most intuitive conception we have of symmetric visuals) and, hence, the size of their representation.

The photographs of real textures also show a four-fold/axis-reflective bias, except that CMM and PMM are more prevalent than in the drawings.

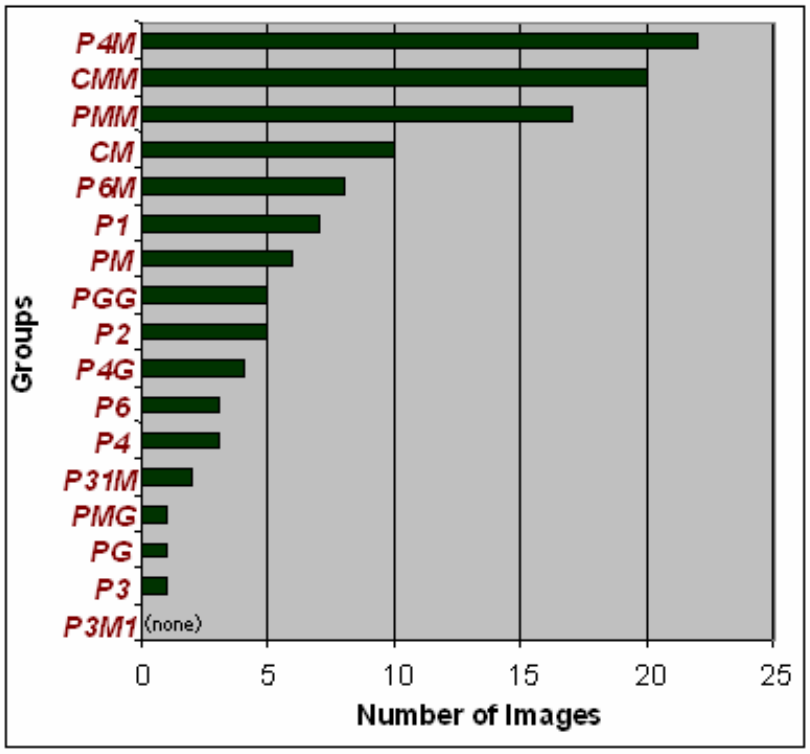


Figure 6.2: Distribution of the ***115 photographs*** in the database

7. Future Work

7.1. Lattice Detection:

The automatic verifier now works by comparing the central tile defined by the lattice with the four tiles laterally adjacent to it, using an SSD comparison score. The limitations of this are similar to the limitations of the original program's symmetry detection – SSD is not a reliable enough indicator of similarity. If the lattice found is wrong but the adjacent tiles thus defined are more or less similar, the verifier might fail to detect the error. Similarly, it could detect an error in an otherwise regular pattern if one or two tiles are irregular.

An improvement would thus be to find a good tile for tile comparison metric that is sensitive enough to detect significant differences. For the second problem, we need an efficient way to compare all the tiles with one another to get a measure of overall regularity.

Another approach that could be examined would be to use something besides autocorrelation to get the peak map. The problem with autocorrelation is that it propagates the translation similarity from the center outward. Thus, the peak map sometimes gets clustered around the center if the tiles all over the image aren't perfectly similar. A procedure that would combine local autocorrelations starting from the center as well as from the edges would be more flexible when dealing with certain images.

7.2. Symmetry Detection

The key goal now is to find a measure of similarity that will reliably give a sharply defined separation between good and bad matches. In the absence of an unambiguous similarity detection procedure, a good clustering method – preferably one that adjusts itself in some way to the score range and variance – is needed.

A fast and reliable Chamfer matching variant would also be useful for the edge image comparison.

For a few images, a wrong symmetry is detected because of loss of color information when they are converted to grayscale (as mentioned in section 3). Implementation of a more robust algorithm for color to intensity conversion would help greatly. We explored some existing methods that claim a better conversion – particularly the recently developed color2gray program presented at SIGGRAPH 2005 [13]. However, as of now, color2gray cannot handle images of the size that are typically processed by our program (the performance is on the scale of $O(S^4)$ for an SxS size image). If a future, more powerful version is developed soon, it could potentially be used for all color to intensity conversions within our program.

7.3. Group Distribution of Pattern Occurrences

A good first step in this area would be to identify relevant research on cognitive perception of textures, in particular, the perception of specific kinds of symmetry. An

examination of this in conjunction with the occurrence distribution across groups of images might yield some explanations about why some groups are more prevalent than others; if there is a neural preference for some symmetry types over others.

It might also be interesting to look at textures defined by natural or engineered objects and see if they correspond to mathematically optimal lattices. A related project would be mapping the symmetry groups to other characteristics of the image and determining if there are any definable relationships.

7.4. Other Ideas

As mentioned earlier, a major project idea would be to look at textures under global transformations – the properties of curvilinear transforms on images in terms of lattices, straightening of patterns using displacement vector detection, and the class of functions mapping points on a straight line to the various curves of real-world image distortions. A further topic would be to explore the relationships between the groups and the pattern properties under distortion.

As mentioned in the source paper [2], the application of symmetry in images to data compression is also an interesting topic. Over the course of this project, I wrote some code that 'compresses' a texture to a single tile and further cuts down the tile on some possible symmetry axes. If the lattice vectors are stored with this cut image, it can then be uncompressed fairly simply. However, this is a relatively primitive way of approaching a data compression question – a more sophisticated method that takes maximum advantage of the lattice packing and group structure of the image would be a great topic for future research.

8. Conclusion

The algorithm used for lattice detection was optimized in time complexity as well as flexibility and accuracy, resulting in a % overall improvement in finding the lattice. The procedures for similarity detection to indicate symmetry were extended to accommodate special classes of images, most notably, images with sparse edges. More similarity detection methods were implemented and score distributions tabulated. The clustering procedure for separating good and bad symmetry scores was sharpened and an extension was added to the group detecting function that would compensate for clustering errors by finding the nearest possible group. These procedures resulted in a % overall improvement in symmetry group identification/estimation. Ideas for future research were also explored and preliminary thoughts were given on pattern identification of textures under global transformations and on the wallpaper group distribution of patterns in art and the real world.

Acknowledgement

The authors would like to thank Sanjeev Koppal for his initial investigation and testing of wallpaper group classification algorithm. We would also like to thank Janice Brothetti for proof reading a draft of this report.

This work is supported in part by an NSF grant IIS-0099597.

References

[1] D. Joyce, Wallpaper Groups (History), http://www.clarku.edu/~djoyce/wallpaper/history.html, 2003

[2] Y. Liu, R.T. Collins and Y. Tsin, "A Computational Model for Periodic Pattern Perception Based on Frieze and Wallpaper Groups," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 3, pp. 354-371, March 2004

[3] Y. Liu, and R.T. Collins, "Skewed Symmetry Groups," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. I872-I879, December 2001.

[4] Y. Tsin, Y. Liu, and V. Ramesh, "Texture Replacement in Real Images," *Computer Vision and Pattern Recognition Conference 2001* (CVPR'01), pp. 539-544, Kauai, December, 2001.

[5] Braddick, O., Campbell, *F*. W., and Atkinson, J. "Channels in vision: Basic aspects." In R. Held, It. W. Leibowitz and H.-L. Teuber (Eds.), *Handbook of sensory physiology (Vol. 8)*, pp. 3-38, Springer-Verlag, New York: 1978.

[6] M.S. Banks, B.J. Stephens, and E.E. Hartmann, "The Development of Basic Mechanisms of Pattern Vision: Spatial Frequency Channels," *Journal of Experimental Child Psychology*, 40, pp. 501- 527, 1985.

[7] H.-C. Lin, L.-L. Wang, and S.-N. Yang, "Extracting Periodicity of a Regular Texture Based on Autocorrelation Functions," *Pattern Recognition Letters*, vol. 18, pp. 433-443, 1997

[8] J. B. A. Maintz, and M.A. Viergever, "A Survey of Medical Image Registration," *Medical Image Analysis*, 2(1), pp. 1-36, 1998.

[9] K. Leszczynski, S. Loose, and P. Dunscombe, "Segmented Chamfer Matching for the Registration of Field borders in Radiotherapy Images," *Physics in Medicine and Biology*, volume 40, no. 1, pp. 83-94, 1995

[10] D.Chetverikov, "Pattern Orientation and Texture Symmetry," in *Computer Analysis of Images and Patterns, Lecture Notes in Computer Science* vol.970, eds. V.Hlavac and R.Sara, Springer Verlag, 1995, pp.222-229

[11] C. Tyler, “The Human Expression of Symmetry: Art and Neuroscience”, http://www.ski.org/CWTyler_lab/CWTyler/Art%20Investigations/Symmetry/Symmetry.html

[12] Y. Liu, W-C Lin, and J.H. Hays, “Near-Regular Texture Analysis and Manipulation,” *ACM Transactions on Graphics (Proc. SIGGRAPH 2004)*, 23(3), pp. 368-376, August 2004

[13] A. Gooch, S. Olsen, J. Tumblin, and B. Gooch, “Color2Gray: Salience Preserving Color Removal,” *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)*, 24(3), pp. 297-307, July 2005