A Wavelet-Based Approach to Template Matching Under Scaling and Translation

Natalie Sauerwald
Advisor: David Meyer
Department of Mathematics
University of California, San Diego

June 6, 2012

Acknowledements

Most importantly, I'd like to thank my advisor, David Meyer. Without his guidance and insight none of this project would have been possible. The amount I have learned throughout this experience goes well beyond what is found on these pages, and I owe much of that to Dr. Meyer. I'd also like to thank the professors and TAs I've had over the past four years, who have challenged and inspired me continually. Coming into college I wasn't one hundred percent sure that I wanted to pursue mathematics, but I quickly realized I had made the right choice after taking a few classes in this department and meeting several of the faculty and graduate students. This department is full of great people, whose advice, enthusiasm, and encouragement in the pursuit of challenges has helped push me towards any success I may have acheived here. Finally, I'd like to thank my friends and family for their continued patience through the past few years. Whether it means listening to my attempts to explain my enthusiasm for a subject few of them care about or simply being there for support through the stressful times, I wouldn't be here without their help.

Contents

Abstract

In this paper we study the problem of template matching in signals, specifically focusing on images. First we present the solution to the case where the template is translated in the image, then discuss how to locate a template if it appears scaled in the image. The tools used for this scaling algorithm are wavelets, specifically the Haar wavelet. Wavelets naturally encode scaling information, which made them attractive for our template matching problem. We discuss using a Hadamard matrix instead of the Haar matrix, because it captures the same information while satisfying some very nice properties. Finally we present our scaling algorithm using this Hadamard matrix, and show some examples and results of this algorithm. In the future this algorithm could be expanded to apply in more cases, and also translated to a quantum setting.

1 Introduction

Template matching refers to a technique of finding which pieces of an image match a given template image. The applications are numerous and extremely diverse, ranging from controlling a robot to medical imaging to computer vision and beyond. This paper in particular is aimed at finding a template matching algorithm which could be easily modified to work in a quantum setting.

The goal of this paper is to present an algorithm for template matching when the image contains a scaled or translated version of the template. Such an algorithm should take inputs of the template and the image, and return the translation amount or the scaling factor. In the case where the template simply appears translated in the image, the answer is well-documented and fairly straightforward. This approach will be explained in the following section. For scaling, however, the problem becomes more complicated. The approach used here is wavelet-based, under the intuition that wavelet transforms give a representation of the input at various resolutions, or scales. More specifically, the Haar wavelet transform is the inspiration for the algorithm, though a Hadamard matrix was used instead for several reasons that will be detailed later in the paper.

The algorithm given at the end of the paper does precisely what was mentioned before, in a special case: the inputs are a template vector and an "image" vector, and the result is the amount of the scaling. We also have an algorithm for a template which is translated and scaled in the image, but while it returns nice results, it is fairly computationally expensive so we are continuing to look for a more optimal algorithm.

1.1 Correlation with Translations

When the template appears exactly somewhere in the image, it is fairly simple to detect the amount of translation using convolution. The idea is that if we convolve the template with the image, the maximum value will occur at the point of correlation, i.e., the translation. To make the computations simpler, we use the convolution theorem. The proof is a standard result which can be found in countless sources, as well as derived fairly easily.

Theorem. (Convolution Theorem) *The Fourier transform of the convolution of f and g is equal to the pointwise product of the Fourier transforms of f and g. Otherwise stated,*

$$f * g = \mathfrak{F}^{-1}[\mathfrak{F}(f) \circ \mathfrak{F}(g)]$$

Proof. The version stated above will be used in the algorithm, but for the proof we will show that $\mathfrak{F}(f * g) = \mathfrak{F}(f) \circ \mathfrak{F}(g)$. Also note that this holds in both the continuous and discrete cases, but as we will be applying it to images, this proof will be of the discrete case. The convolution of f and g is given by $f * g = \sum f[x]g[z - x]$, where f and g are discrete functions, so

$$\begin{aligned}\mathfrak{F}(f * g) &= \sum_z \sum_x f[x]g[z - x]e^{-2\pi izv} \\ &= \sum_x \sum_z f[x]g[z - x]e^{-2\pi izv}\end{aligned}$$

Substituting $y = z - x$ gives

$$\begin{aligned}&= \sum_x f[x] \sum_y g[y]e^{-2\pi i(y+x)v} \\ &= \sum_x f[x]e^{-2\pi ixv} \sum_y g[y]e^{-2\pi iyv} \\ &= \sum_x f[x]e^{-2\pi ixv} \sum_y g[y]e^{-2\pi iyv} \\ &= \mathfrak{F}(f) \circ \mathfrak{F}(g)\end{aligned}$$

□

Using this, the algorithm for template matching is very straightforward. If we take the inverse Fourier transform of the pointwise product of the Fourier transform of the template and the image, the maximum value will appear at the point desired. In image processing we use the discrete, two dimensional Fourier transform, because both the template and the image are matrices of discrete values. If t is the template and v is the image, the algorithm finds the maximum of

$$\mathfrak{F}^{-1}[\mathfrak{F}(t) \circ \mathfrak{F}(v)].$$

An example of the results of this algorithm is shown in **Figure 1**. The coordinates at the bottom of the third image give the point where the top right corner of the template lines up with the image. Though the template appears to be much larger than the version in the image, it is in fact an exact copy of the pixels that exist in the original image.

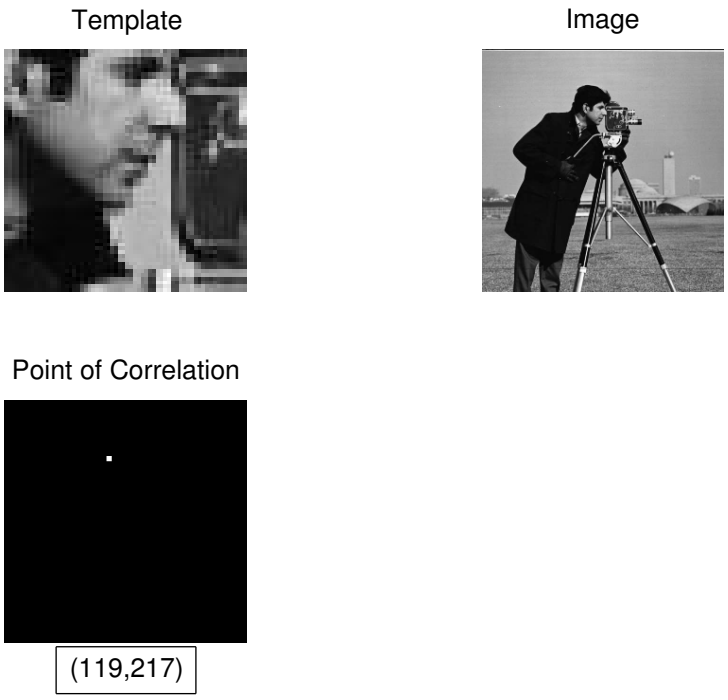

Figure 1: Example of Correlation Algorithm

2 Wavelets

Wavelets are a tool often used in image processing to study an image at various scales, making it an appropriate choice for our problem here. Wavelet analysis is essentially an extension of Fourier analysis which studies the coefficients of wavelets instead of those of sines and cosines. Similar to the Fourier transform, wavelets can be used to reconstruct a signal entirely by adding together and manipulating the terms. By nature, wavelets encode the changes in a signal at many scales, allowing us to see overall trends as well as finer resolution changes. A locally constant signal makes certain wavelet coefficients of zero, which generally leads to sparse results. For this reason, wavelets are frequently used in image compression and are the basis for the JPEG 2000 standard[3]. The choice of wavelets can be adapted to a specific problem, though there seems to be debate on the topic of how to chose a type of wavelet, and whether it is worth spending time choosing any particular type[2].

The scaling nature of wavelets suggests that they should be a good choice for this problem. If the template appears scaled in the image, it should show the same pattern of changes at some level of detail of the wavelet transform. The idea is that the wavelet transform of the image and that of the template will match up at some level, and we will be able to detect that and discern the scaling factor based on where they match up[1].

2.1 The Haar Wavelet

The type of wavelet studied for this paper was the Haar wavelet, an orthogonal wavelet transform which uses a combination of averages and differences to study the details of a signal at many levels. Wavelets can be described by a mother wavelet function $\psi(t)$ and a scaling function, $\phi(t)$. The Haar wavelet is therefore given by

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$

In the Haar case, the scaling function takes averages over certain intervals, while the wavelet function encodes the differences between adjacent values

of the signal[2]. As an illustration of the 8×8 case, the Haar matrix looks like this:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

This matrix demonstrates several important properties of the Haar wavelet. One thing to note is that the columns are all orthogonal, and can be normalized so that they are orthonormal. Not all wavelets are orthogonal, but the Haar wavelet belongs to the class which is. It is also clear from this matrix that the result of multiplying this matrix with a given vector will tell us a variety of information about the original vector. The first entry gives the average— the broadest possible view of a signal. The next entry is a slightly finer scale view, showing the difference of the average of the first half of the signal with the second half. Each subsequent entry in the resultant vector gives a finer scale view of the original signal, with the last four showing any small changes or lack thereof. For most images, the result of the Haar transform will be sparse, given that in an image most adjacent pixels have similar values, except for changes such as edges or lighting. One of the strengths of wavelets is the fact that not only can we analyze much of the information in a signal from the results of the wavelet transform, but the original signal can be entirely reconstructed from just this information because the Haar matrix is invertible[4].

2.2 The Hadamard Matrix

While the Haar transform has the properties we are looking for to detect scaling, it is a bit difficult to work with as a matrix. In just the 8×8 case, we need three different normalizing factors for the different rows. There is no simple formula for determining each value in the matrix, rather it follows a pattern which is clear to a viewer but harder to program into a computer and access without a relatively complicated formula. For these reasons, and also with an eye to translating this algorithm into a quantum setting, we decided to use the Hadamard matrix instead. Each entry is determined by the simple formula

$$H_{i,j} = (-1)^{i \cdot j}$$

where i and j are binary strings, and the labelling of rows and columns begins with zero. For comparison, we can look at the 8×8 version:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

Note that in order to normalize this matrix we can simply multiply the entire matrix by a factor of $\frac{1}{\sqrt{n}}$, in this case $\frac{1}{2\sqrt{2}}$. It also is its own inverse (once normalized), which is very convenient in minimizing computations. It retains the property of all of its rows being mutually orthogonal, and can be constructed through basic row operations on the Haar matrix. The Hadamard matrix is simply a linear transformation of the Haar matrix, which shows that the Hadamard matrix encodes the exact same information we get from the Haar matrix. For all of these reasons the Hadamard matrix appears to be a better choice for our algorithm, while giving us the same information we were looking for in choosing the Haar transform.

3 Scaling Algorithm

Using this Hadamard matrix we can construct a scaling algorithm which is fairly similar in form to the original convolution algorithm which we used in the simple scaling case. We have reduced our problem from images with pixel values of zero to 255 down to vectors of ± 1. This was accomplished by computing the average pixel value of the image, then setting any pixels greater than this average equal to 1, and all pixels with value less than the average were set to -1. This creates a kind of silhouette of the original image, maintaining the overall shapes. So the input template has entries of only ± 1, padded by zeros to be the same size as the "image" vector. We only consider scalings by a factor of 2^n, given that the scales encoded by the Haar wavelet are only powers of two. There is one last thing to mention before we discuss the algorithm itself.

3.1 S Permutation Matrix

One important part of the algorithm is an operation which will be referred to as S. S is a permutation matrix which takes a binary string and moves the last bit to the front, permuting the vector entries or matrix rows accordingly. By extension, S^k cyclically moves the last k bits of the string to the front.

Example. *Below we see how S reorganizes the entries of a vector. The leftmost vector is the original v, the middle vector shows Sv, and the last shows S^2v.*

$$\begin{pmatrix}0\\1\\2\\3\\4\\5\\6\\7\end{pmatrix} \longrightarrow \begin{pmatrix}0\\2\\4\\6\\1\\3\\5\\7\end{pmatrix} \longrightarrow \begin{pmatrix}0\\4\\1\\5\\2\\6\\3\\7\end{pmatrix}$$

Note in particular that S^k moves the entries which are multiples of 2^k up to the top (with labelling starting at zero), and the rest of the entries down to the bottom. This will be important later— indeed it is the entire reason for using this S permutation in our algorithm.

3.2 The Actual Scaling Algorithm

The result of this algorithm is a matrix, taking inputs t and v, the template and the scaled version, and returning a matrix in which the k^{th} column represents a scaling by 2^k. Algebraically, we define the n^{th} entry of the scaled version (scaled by k) to be the $\lceil \frac{n}{k} \rceil$ entry of the template. As an example, the vector on the left could be a template, and the vector on the right represents the template scaled by two.

$$\begin{pmatrix}4 & 7 & 8 & 2 & 0 & 0 & 0 & 0\end{pmatrix} \to \begin{pmatrix}4 & 4 & 7 & 7 & 8 & 8 & 2 & 2\end{pmatrix}$$

The result of the scaling algorithm is a matrix where each column w_i is given by:

$$w_i = H(Ht \circ S^i Hv)$$

where $\circ$ represents a pointwise product. We could even reduce this to only consider the first row of this result, further minimizing computations by only multiplying by the first row of H, instead of the entire matrix.

Claim. *The maximum value of the first row of this matrix will occur in the column which represents the correct scaling.*

Proof. We will show that if v is scaled by 2^k,

$$\vec{1} \cdot (Ht \circ S^k Hv) \geq \vec{1} \cdot (Ht \circ S^j Hv)$$

where $\vec{1}$ represents a vector of all ones of the appropriate size, and $\circ$ is the dot product. Note that this is the first row of the Hadamard matrix, so this is equivalent to taking the first row of the matrix originally described above. First we will consider the vector Ht. If t has length n and v is scaled by 2^k, t should have $l = \frac{n}{2^k}$ nonzero terms at the top, followed by a padding of zeros. These zeros essentially eliminate the last $n - l$ columns of H when considering Ht, so we will now consider the structure of the first l columns of H. Recall that the entries of H are defined by the formula

$$H_{i,j} = (-1)^{i \cdot j}$$

where i and j are the labels in binary, with the first row and column labeled as zero. Note that because l is a power of 2 and we are only considering $j < l$, $(i + l) \cdot j = i \cdot j$, so $H_{i,j} = H_{i+l,j}$. Because of this, Ht also has the property that $Ht_i = Ht_{i+l}$ where the subscript indicates the entry in the vector. For simplicity later on, let

$$Ht = \begin{pmatrix} X \\ X \\ \vdots \\ X \end{pmatrix} \quad \text{where} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{pmatrix}$$

and there are 2^k copies of X in Ht. Now we will look at the structure of Hv. We defined v to be t, scaled by 2^k. This means that the first 2^k entries of v are equal, as are the second 2^k entries, and so on. Because of this, all entries of Hv which are not multiples of 2^k (when the first row and column are labelled as zero) will be zero. For all of the odd rows, $H_{i,j_{even}} = -H_{i,j+1}$, so given that $v_{i_{even}} = v_{i+1}$ with any scaling, all of these rows will go to zero. The same argument applies for larger scaling factors, so that Hv has the

basic structure

$$\begin{pmatrix} v_1 \\ 0 \\ \vdots \\ 0 \\ v_2 \\ 0 \\ \vdots \\ 0 \\ v_3 \\ \vdots \end{pmatrix}$$

with each set of zeros containing $2^k - 1$ zeros, so that the nonzero terms all occur at multiples of 2^k. Now we would like to look more closely at what the nonzero terms actually are. They correspond to the dot product of the rows of H which are multiples of 2^k with the vector v. So these terms are equal to $H_{m2^k,j} \cdot v$ for some $m \in \mathbb{N}$. Note that within a row which is a multiple of 2^k, we get that

$$H_{i,m2^k} = H_{i,m2^k+1} = \cdots = H_{i,m2^{k+1}-1}$$

Starting with the most basic case, it is fairly clear that the first entry in Hv should be $2^k x_1$, or 2^k times the first entry in Ht. Similarly, the second nonzero entry in Hv, at position 2^k, is $2^k x_2$. So in fact we have a good idea of what Hv looks like:

$$\begin{pmatrix} 2^k x_1 \\ 0 \\ \vdots \\ 0 \\ 2^k x_2 \\ 0 \\ \vdots \\ 0 \\ 2^k x_3 \\ 0 \\ \vdots \end{pmatrix}$$

This is where we use the S matrix described earlier. S was constructed such

that

$$S^k Hv = \begin{pmatrix} 2^k X \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Any other power of S will permute the entries of Hv, but will not give the same result. Now when we note that for any two vectors u and v

$$\vec{1} \cdot (u \circ v) = u \cdot v$$

where again $\vec{1}$ is the vector of all ones, $\cdot$ refers to the dot product, and $\circ$ is the pointwise product. Therefore we are now trying to show that $(Ht \cdot S^k Hv) \geq (Ht \cdot S^j Hv)$ for all k and j. Given our previous work,

$$Ht \cdot S^k Hv = \begin{pmatrix} X \\ X \\ \vdots \\ X \end{pmatrix} \cdot \begin{pmatrix} 2^k X \\ 0 \\ \vdots \\ 0 \end{pmatrix} = X \cdot 2^k X = 2^k (X \cdot X)$$

For $j \neq k$, $Ht \cdot S^j Hv$ will be the same length as $Ht \cdot S^k Hv$, but it will be the sum of cross terms instead of the sum of the squared terms, so by the Cauchy-Schwarz inequality, we have that $Ht \cdot S^k Hv \geq Ht \cdot S^j Hv$, as desired. Note that Cauchy-Schwarz also asserts that these will be equal only when $S^k Hv = S^j Hv$, so we are assured to have a unique maximum. □

3.3 Matching with Imperfections

Often in template matching the template is not exactly represented in the image, but there are a few slight differences or imperfections in the copy. In an image this might be a result of noise, imperfections in the image quality, change in lighting, or a number of other things. Algebraically this means that the pixel values may not match up exactly. With our technique of thresholding the image at its average pixel value and converting all pixels above that threshold to positive one and all pixels below the average to negative one, we already account for small changes in pixel values. In most cases, if a pixel in our template had a grayscale value of 203 for example, but in the image the matching pixel(s) had a value of 209, or some other value close to our original pixel, both of these will be sent to positive one, so they will still be identical when plugged into the algorithm. In the case of noise or a change that happens to be close enough to the average where a pixel

in the template will be sent to its negation in the image, we end up with an image vector which is very close to a scaled version of our template, but with a few very small imperfections. The algorithm should still return the correct scaling factor here, though perhaps the maximum will not be quite as separated from the other values as in the case of a perfect copy. We can see in the proof of our algorithm that we will still be comparing the dot product of two very similar vectors against the dot product of two less similar vectors. Though the best results clearly come from the template being copied exactly in the image, this scaling algorithm is strong enough to still capture the correct scale, even given a few imperfections in the matching.

3.4 Examples and Results

First we will show an explicit example for the 8×8 case, then I will show a few results for larger inputs using the program I wrote for this algorithm, including an example where the image is not an exact copy of the template, but contains some imperfections.

Example. *In the 8x8 case, we will consider a template with four nonzero entries, and the scaled version as twice the template. So we have*

$$t = \begin{pmatrix} -1 \\ 1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad v = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

Multiplying both by H yields the following

$$Ht = \begin{pmatrix} -2 \\ -2 \\ 2 \\ -2 \\ -2 \\ -2 \\ 2 \\ -2 \end{pmatrix} \qquad Hv = \begin{pmatrix} -4 \\ 0 \\ -4 \\ 0 \\ 4 \\ 0 \\ -4 \\ 0 \end{pmatrix}$$

So if we apply S to Hv and take the dot product, we get

$$\vec{1} \cdot (Ht \circ SHv) = \begin{pmatrix} -2 \\ -2 \\ 2 \\ -2 \\ -2 \\ -2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} -4 \\ -4 \\ 4 \\ -4 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 32$$

In contrast, if we apply S^2 to Hv, instead we have

$$\vec{1} \cdot (Ht \circ S^2Hv) = \begin{pmatrix} -2 \\ -2 \\ 2 \\ -2 \\ -2 \\ -2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} -4 \\ 4 \\ 0 \\ 0 \\ -4 \\ -4 \\ 0 \\ 0 \end{pmatrix} = 16$$

So as desired the maximum (in this case 32) occured when we used S instead of S^2, because our vector was scaled by 2 instead of 4.

Example. *We have only seen the results for a very small input with very limited scaling possibilities. For much larger inputs we prefer not to do the computations by hand, so the results shown here are the results of a program written to implement the algorithm. In the first example, the input vectors have length 64, and the nonzero entries of the template are shown below. v is scaled by a factor of 8 in this case.*

$$t = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

The algorithm gives a resulting row vector shown below, where the maximum value shows us the correct scaling factor.

$$\begin{pmatrix} -128 & 384 & -384 & \boldsymbol{512} & -128 & 128 \end{pmatrix}$$

The maximum here is clearly 512, and it occurs in column three (remember we begin labelling by zero), so the scaling factor should be $2^3 = 8$.

Example. *In the next example, the inputs have length 128 and the template has four nonzero values, therefore it has been scaled by 32. The nonzero values are again given below*

$$t = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

This time the result we get is given by

$$\begin{pmatrix} 256 & 256 & 256 & 256 & 256 & \boldsymbol{512} & 256 \end{pmatrix}$$

Again the maximum is 512 and this time it occurs in column 5, so the scaling must have been by $2^5 = 32$.

Example. *Here we will show an example of template matching with imperfections in the image vector. The inputs have lengths of 16, and the template will be scaled by 4 in the image. However, three of the entries in the scaled version have been negated to simulate an imperfect scaling. These imperfections were randomly chosen, and happen in the second, sixth, and sixteenth entries. The template and scaled version with imperfections are shown below.*

$$t = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad v = \begin{pmatrix} -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

For these inputs the algorithm returns the following vector:

$$\begin{pmatrix} 0 & 32 & \boldsymbol{64} & 0 \end{pmatrix}$$

The maximum, 64, occurs in the second column demonstrating a scaling of 4 despite the imperfections.

Example. Figure 2 *shows the output of the scaling algorithm applied to an image. In this case the inputs were clearly matrices with color pixel values, so it was first converted to grayscale. At that point the values ranged from zero to 255, so we then reduced the image to* ± 1 *using the thresholding concept described earlier. The algorithm in this case had to be adapted to work for matrices instead of just vectors, but the concept remains the same. The program used to find these results can be found in the appendix. The vector output by the program is:*

$$\begin{pmatrix} 1536 & \boldsymbol{10752} & -2560 & -6656 & -5632 & 2560 & -3584 & 1536 \end{pmatrix}$$

The maximum is 10752, occuring in column 1 so the image is scaled by 2.

3.5 Preliminary Translation and Scaling Algorithm

Unfortunately, it is not often the case that in template matching we are looking either for a translation or a scaling— often both occur together. The algorithm given for scaling does not seem to be able to capture both, so we have a preliminary algorithm which has that capability, though it is fairly inefficient. It is based entirely on the previous algorithm, and only works for shifts of 2^k. The idea is that instead of outputting a vector, the output will be a matrix in which the columns still represent the different scaling factors and the rows represent the different translations. In this case the location of the maximum tells the exact translation amount and scaling amount, with the first row representing no translation (instead of a translation of 2^0), and for all other rows, row i represents a translation of 2^i. We essentially compute the above algorithm for each given translation, and ouput them as described.

Example. *In this example the inputs both have length 32, and the template has 8 nonzero values. The translated and scaled version has been shifted by 8 and scaled by 4. The nonzero entries of the template and the results of the*

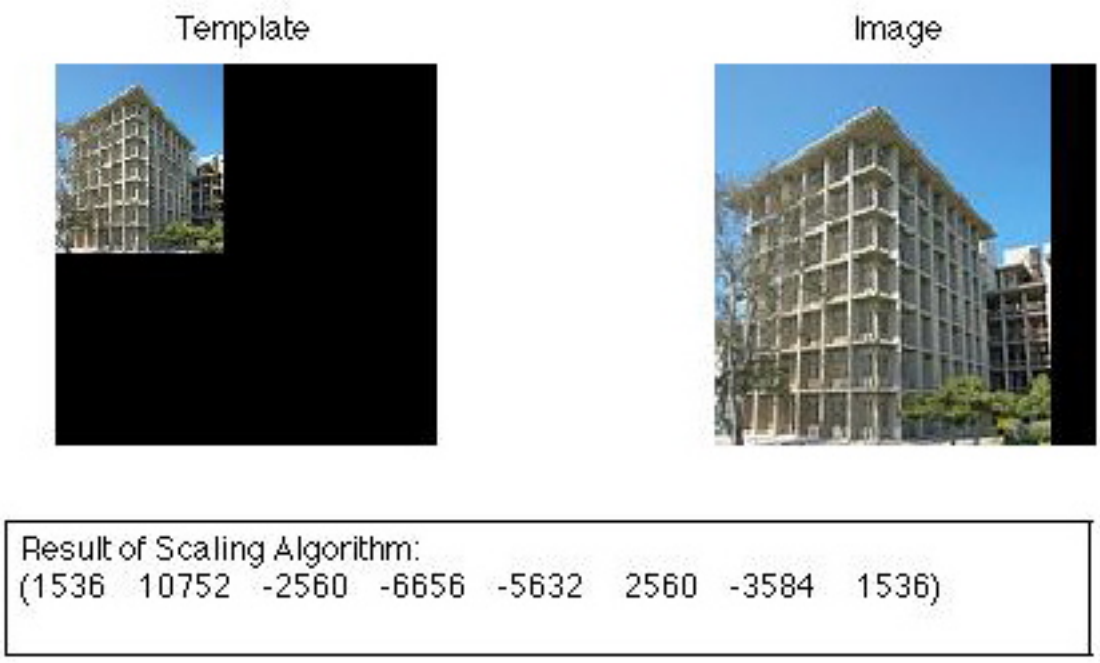

Figure 2: Example of scaling algorithm applied to an image

matrix are shown below.

$$t = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 64 & 64 & -128 & 192 & 64 \\ 64 & -64 & -128 & 192 & 64 \\ -64 & -64 & 0 & -64 & -64 \\ 64 & 64 & \boldsymbol{256} & -64 & 64 \\ 64 & 64 & -128 & -64 & 64 \end{pmatrix}$$

The maximum is 256, and it occurs in column 2, representing a scaling of $2^2 = 4$ *and in row 3, representing a translation of* $2^3 = 8$ *as expected.*

4 Conclusions

As we have demonstrated, the Haar transform does provide an algorithm which is analogous to the solution to the translation problem mentioned in the introduction. Taking advantage of the scaling nature of wavelets, we were able to design an approach to template matching under scales of 2^k which always correctly identifies the scale, even if the image is not exactly identical to the template. The Hadamard matrix provided a nicer basis than the Haar matrix given that it is symmetric and unitary, can be scaled with only one factor, has a simple formula for determining its entries, and is more easily translated for a quantum algorithm. It was clearly able to encode the same information as the Haar matrix, and gave us very nice results.

Future work on this project includes finalizing the translation and scaling algorithm, hopefully finding a way to make it more efficient. We also would like to translate this to a quantum setting. I would also like to find a way to detect translations and scalings which are not powers of 2, but it appears that the Haar wavelet may not be the right way to do that, since it very effectively captures scales which are powers of 2, but is much less effective at any other scale. Wavelets still seem to be an appropriate tool, but the Haar wavelet seems to be too simple and idealized only for scales of two. Beyond this there are countless other possibilities for continuing this work, including detection of rotated templates or templates which have been degraded at some point. Some of these questions have been studied already, but there is room for a huge amount of research to continue in this field of template matching.

5 Appendix – MATLAB code

This appendix contains all of the relevant programs I have written for this project (all in MATLAB).

1. Correlation (translation only case)

```
function out = imagecorrelation2(A,B)

%Finds the point of correlation between a template and an image
%input order doesn't matter- template or image could be first

[m1,n1,k1] = size(A);
[m2,n2,k2] = size(B);

%In case the input images are in color, we convert them to black&white
if k1>1
    A=rgb2gray(A);
end
if k2>1
    B=rgb2gray(B);
end

%The following group of code normalizes the pixel to all be roots of unity
%Without this step the program won't find the correct correlation
C=zeros(1,257);
C(1,1)=1;
C(1,257)=-1;
w=roots(C);
A2=zeros(m1,n1);
B2=zeros(m2,n2);
for i=1:m1
    for j=1:n1
        A2(i,j)=w(A(i,j)+1);
    end
end
for i=1:m2
    for j=1:n2
        B2(i,j)=w(B(i,j)+1);
    end
end
```

```
%Here is the one line which implements the Correlation Theorem
X= ifft2((conj(fft2(A2, m2, n2)).*fft2(B2)));

%We find the maximum and output the indices at which it occurs
[x,y]=max(X(:));
[i,j]=ind2sub(size(X), y)

%This simply makes the point of correlation easier to visually see- now it
%is a 10x10 box of white instead of only 1 pixel
corr=zeros(size(X));
for a=i:i+10
    for b=j:j+10
        corr(a,b)=1;
    end
end

%We plot the inputs and results
subplot(2,2,1), imshow(A);
title('Template');
subplot(2,2,2), imshow(B);
title('Image');
subplot(2,2,3), imshow(corr);
title('Point of Correlation')
```

2. Hadamard Matrix

```
function H = hadamard(L)

%Creates an nxn Hademard matrix, where n is the smallest power of 2
%still greater than the input L

n=ceil(log2(L));

for i=1:2^n
    for j=1:2^n
        H(i,j)=(-1)^(de2bi(i-1,n)*de2bi(j-1,n)');
    end
end
```

```
%The line below can be uncommented to create an orthonormal Hadamard matrix
%H = (1/sqrt(2^n))*H;
```

3. S Permutation Matrix

```
function S = spermutation(n,m)

S=zeros(n,m);

for i=1:n
    for j=1:m
        a=zeros(log2(n),1);
        a(1:ceil(log2(i)))=de2bi(i-1);
        b=zeros(log2(n),1);
        b(1:ceil(log2(j)))=de2bi(j-1);
        if a==circshift(b,-1)
            S(i,j)=1;
        end
    end
end
```

4. Scaling Algorithm

```
function H = htransform2 (temp,new)

[n,m]=size(temp);

Ht=hadamard(n)*temp;

Hv=hadamard(n)*new;

%The vectors x and b created below are needed for the S permutation matrix,
%and adjustments of +/- 1 have to be made for the indices because Matlab
%labels vectors and matrices beginning with 1 instead of 0, as the
%algorithm dictates.

x=zeros(n,log2(n));

for i=1:n
```

```
    for j=1:log2(n)
        s=(spermutation(n,n))^(j-1);
        sHv=s*Hv;
        H(i,j) = (Ht(i,1))*(sHv(i,1));
    end
end

%The line below gives the same result as taking the first row of the result
%of using the inverse Hadamard matrix
H=ones(1,n)*H;
```

5. Converting Images to 0,1

```
function X = convertbw(A)

[n,m,k] = size(A);

%First we convert the image to grayscale if it was in color
if k>1
    A=rgb2gray(A);
end

%Take the treshold to be the average pixel value
thresh=mean(A(:));

X=zeros(n,m);

%For all pixels above the treshold, set them equal to 1, and all below
%the threshold set to 0
for i=1:n
    for j=1:m
        if A(i,j)>=thresh
            X(i,j)=1;
        else
            X(i,j)=0;
        end
    end
end
```

6. Scaling Algorithm Adapted for Images

```
function H = himage(temp, image)

[n1,m1,k1] = size(temp);
[n2,m2,k2] = size(image);


%First change the inputs to have values of +/-1 instead of 1s and0 0s
for i=1:n1
    for j=1:m1
        if temp(i,j)==0
            temp(i,j)=-1;
        end
        if image(i,j)==0
            image(i,j)=-1;
        end
    end
end

%The following is the implementation of the algorithm- to adapt it to
%images we have to multiply S by the transpose of the matrix twice
%essentially, so we look at S[((S^j)Hv')'] instead of just (S^j)Hv
Ht=hadamard(n1)*temp;
Hv=hadamard(n1)*image;

for j=1:log2(n1)
        s=(spermutation(n1,n1))^(j-1);
        sHv=(spermutation(n1,n1)*(s*Hv)')';
        H(1:n1,j) = (ones(1,n1)*(Ht.*sHv))';
end
```

7. Scaling and Translation Algorithm

```
function H = haarshsc (temp,new)

%Shifting and Scaling algorithm- temp should be the template, and "new" is
%the result of shifting and scaling the template by some power of 2

[n,m]=size(temp);
```

```
Ht=hadamard(n)*temp;

x=zeros(n,log2(n));

%The first for loop runs the algorithm over every possible shift of 2^k,
%and the second and third for loop are copied almost exactly from the
%htransform2.m file
for k=1:log2(n)
    if k<2
        Hv=hadamard(n)*new;
    else
        Hv=hadamard(n)*(circshift(new, -2^(k-1)));
    end

    for i=1:n
        for j=1:log2(n)
            b=zeros(log2(n),1);
            b(1:ceil(log2(i)))=de2bi(i-1);
            x(i,j)= (Ht(i,1)*Hv(bi2de(circshift(b',[1,j-1]))+1,1));
            H(k,1:log2(n))=ones(1,n)*x;
        end
    end
end
```

6 References

[1] Graps, Amara, 1995: An Introduction to Wavelets. *IEEE Computational Science & Engineering*, **1070-9924**, 50-61.

[2] Hubbard, Barbara Burke, *The World According to Wavelets*. A K Peters, Wellesley, Massachusetts, 2nd Edition, 1998.

[3] Rabbani, Majid, and Rajan Joshi, 2002: An Overview of the JPEG 2000 Still Image Compression Standard. *Signal Processing: Image Communication*, **17-1**, 3-48.

[4] Walnut, David F., *An Introduction to Wavelet Analysis* Birkhäuser Boston, 2004