Compressive Image Feature Extraction by Means of Folding

by

Brian Calvin Gardiner

S.B., Massachusetts Institute of Technology (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author .....
Department of Electrical Engineering and Computer Science
May 15, 2012

Certified by .....
Dr. Christopher Yu
Sensors & Networks Group Leader, Draper Laboratory
Thesis Supervisor

Certified by .....
Prof. Piotr Indyk
Professor, MIT
Thesis Supervisor

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Compressive Image Feature Extraction by Means of Folding

by

Brian Calvin Gardiner

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We explore the utility of a dimensionality reducing process we term *folding* for the purposes of image feature extraction. We seek to discover whether image features are preserved under this process and how to efficiently extract them. The application is in size weight and power constrained imaging scenarios where an efficient implementation of this dimensionality reduction can save power and computation costs.

The specific features we explore are image corners, rotation, and translation. We present algorithms for recovering these features from *folded* representations of images followed by simulation results showing the performance of the algorithms when operating on real images.

Thesis Supervisor: Dr. Christopher Yu
Title: Sensors & Networks Group Leader, Draper Laboratory

Thesis Supervisor: Prof. Piotr Indyk
Title: Professor, MIT

Acknowledgments

I first and foremost thank my advisors, Chris Yu and Piotr Indyk, who guided my research. My weekly meetings with them were invaluable in steering my work down a productive path. I also thank Chris for opening up the DLF position to me through the 6-A program, so that I could work at Draper Laboratory for the year.

Alan Oppenheim provided much helpful insight towards the beginning of the project and for that I thank him.

I thank Draper Laboratory for funding my studies and for providing me a productive environment to carry out research. Many conversations with officemates and other lab members proved fruitful in developing this work.

I lastly thank my parents and brother for helping me to get to where I am today, my girlfriend Brittany for the support she has given me these past 5 years at MIT, and my close friends for providing much needed fun distractions from my work.

Contents

List of Figures

Chapter 1

Introduction

In this thesis we explore the preservation of a number of different image features under a dimensionality reducing process we term *folding*. The features explored are image corners, rotation, and translation. A series of algorithms for recovering these features from folds are presented along with simulation results. Following is some preliminary background discussion.

1.1 Folding

The dominant theme of this thesis is acquiring features of an image from its folded representation. This can be viewed as a particular manifestation of the common problem of acquiring characteristics about a signal from an under-sampled or lower dimensional representation. For this thesis, the signals are two-dimensional images, and the undersampling process to create the lower-dimensional representations is folding.

In order to proceed, we must clearly define what is meant by folding; it can be a rather ambiguous term. For the purposes of this thesis, the same folding process from [12] will be used. It should be thought of as a superposition of non-overlapping subsections of the original image. Formally, if $I[x_1, x_2]$ is the input image, the output

from folding the image by p_1 in the first dimension and p_2 in the second dimension is:

$$\texttt{FOLD}(I, p_1, p_2) = m[y_1, y_2] = \sum_{\substack{y_1 \equiv x_1 \pmod{p_1} \\ y_2 \equiv x_2 \pmod{p_2}}} I[x_1, x_2]$$

Figure 1-1 illustrates the folding process used in this thesis. It should be noted that the horizontal and vertical orientation flipping that would occur in folding a piece of paper do not occur in this folding process.

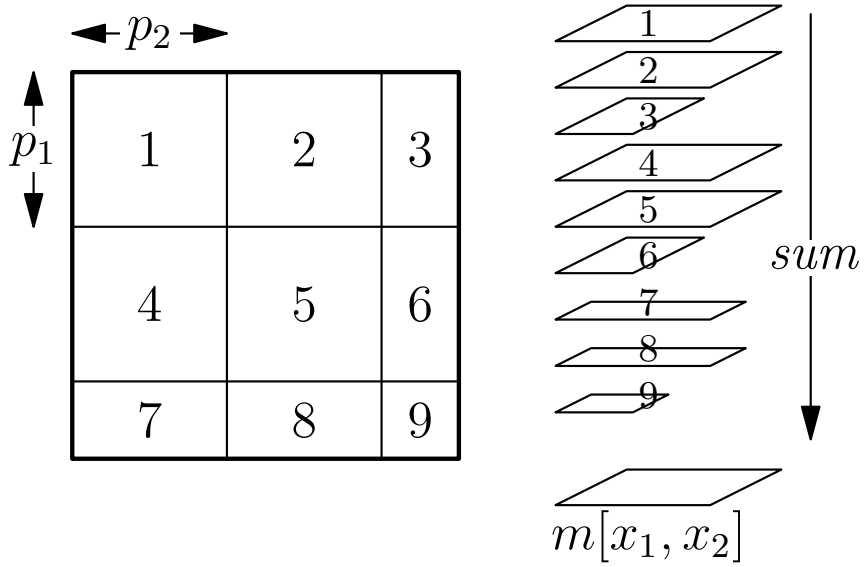


Figure 1-1: The folding process used in this thesis.[12].

Folding offers a number of advantages as a method of undersampling. One, it tends to preserve local features. This was observed in [12] where it was shown that folding preserves local geometric references of star positions in night sky images. Two of the three image properties explored in this work, corners and gradient direction, are local in nature. Second, folding is amenable to a hardware implementation.

1.2 Motivation

Most current-day digital imaging architectures work as described in Figure 1-2. The scene is focused via an optical lens onto the focal plane array, typically using either CCD or CMOS technology. Analog to digital conversion occurs to convert image

intensity values from the focal plane into digital pixel values. For most applications, the image data is then compressed using one of the many image coding methods (i.e. JPEG). The problem with this architecture that we address is that there is considerable wasted power in the image acquisition phase. Many of the compression techniques used are lossy and so a significant amount of power is used to process data in the acquisition phase only to be immediately discarded in the compression phase.

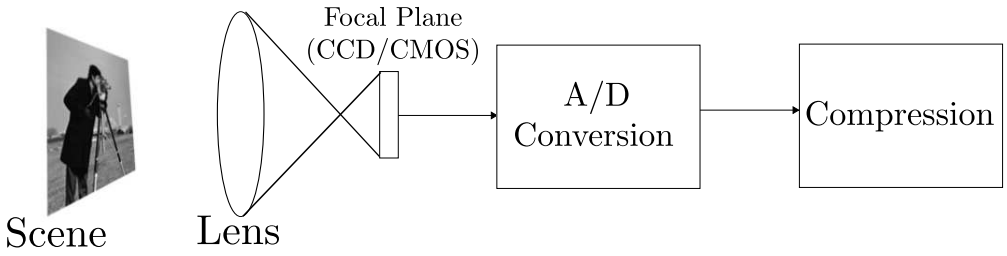


Figure 1-2: The typical design of current imaging architectures.

This architecture is less than ideal in numerous settings particularly where size, weight, and power are constrained. The ideal imager in these settings is as small as possible, as light as possible, and consumes as little power as possible. Moreover, the majority of power consumption in imagers is due to the A/D conversions. The wasted power between the digitization step and the compression step in Figure 1-2 is very costly. This begs the question of whether it is possible to somehow directly acquire (digitization phase) the information in a compressed representation (compression phase). This is the underlying idea of compressive sensing [8],[5].

As an illustration of this inefficiency, consider a small unmanned aerial vehicle (UAV) in a GPS constrained environment. An approach in such a setting is to rely on visual imagery for navigation purposes. This involves some sort of image feature tracking from frame to frame. The inefficiency with current imaging architectures is that the information rate, the feature information, is usually much less than the data rate, the digitized values at every pixel location. In this thesis we present a system that reduces the data rate closer to the information rate by A/D converting fewer aggregate image measurements than a standard imager while still extracting the desired image properties. This achieves the desired result of pushing some of the

compression phase in Figure 1-2 into the A/D conversion phase.

1.3 Previous Work

1.3.1 Folding

This thesis builds on the ideas established in [12],[11]. In these papers, the authors introduce the concept of folding as a mechanism for recovering local geometric features from an image. Rigorous theoretical analysis of the techniques is described as well as results from applying the algorithms to star images. A specific application of attitude determination from a captured image of the night sky in space is developed. The ideas of folding as well as Chinese Remainder Theorem decoding are the the main results that serve as the basis for this thesis. The significant departure in this thesis is exploring images that do not exhibit spatial sparsity. When the input image is spatially sparse, that is most pixels have a value of zero, folding introduces relatively little degradation as most non-zero pixels end up being summed with zero-valued pixels in the folded representation. Such is the case with star images of the night sky. Extending the class of input images to natural images poses a challenge for the folding architecture. The zero-valued pixels of spatially sparse images become non-zero for natural images. In this case folding introduces *distortion* in the form of 'signal noise' as features of interest are added to other non-zero pixel values.

1.3.2 Compressive Sensing

Compressive sensing, introduced in [8] and [5], is a recent direction in signal processing that examines how to reconstruct a signal from a lower-dimensional sketch under the assumption the signal is sparse. The problem can be stated as given an arbitrary input vector x, finding a matrix A and a corresponding recovery algorithm to find a vector $\hat{x}$ from the lower-dimensional sketch Ax such that:

$$\|x - \hat{x}\|_1 \leq C \cdot Err_k^1(x)$$

where $Err_k^1(x)$ is the minimum L-1 error between x and x', $\|x - x'\|_1$, over all k-sparse vectors x'. Other L_p norms have also been investigated and substantial research has been conducted recently about what measurement matrices A and recovery algorithms are optimal depending on the application [10], [22].

This thesis shares some common goals with the field of compressive sensing. Namely, we are trying to garner information about a signal from a lower-dimensional sketch. In fact, folding is a linear operation and just one specific type of measurement matrix A for taking a sketch of an image. If a two-dimensional image is represented as a single vector, by concatenating all of the columns on top of each other, then folding can be represented as a simple matrix multiplication Ax, where A is a relatively sparse binary matrix with a particular structure corresponding to the folding process. On the contrary, there are some key differences that separate this work from the standard class of compressive sensing problems.

- **Feature Recovery**. Current compressive sensing research in regards to images is recovery-focused. There is a wide body of literature on which sketching matrix A and corresponding recovery algorithm are optimal for images, however they are focused on recovering the original image, or at least an approximation. This work differs in that we never attempt to recover the original image from our folds but are rather solely interested in feature extraction from the images. The recovery algorithms presented thus differ significantly with the L_p-minimization methods typical of compressive sensing techniques.

- **Hardware Implementable Solutions**. The design space for the compressive feature detection algorithms presented in this thesis only includes solutions that have a design path to straight-forward hardware implementations. Power savings from implementing such a design in hardware is a primary motivation for this research. Structured binning or folding in the focal plane array may lend itself to efficient hardware implementations, and consequently it was chosen as the form of undersampling for exploration in this thesis. On the contrary, most of the research in compressive sensing is theoretical in nature, with implemen-

tation considerations largely not addressed. Many of the sketching matrices, A, that are ideal for recovery from a theoretical perspective, such as random Bernoulli or Gaussian matrices, do not have straight forward hardware implementations. Moreover, many of those solutions that do exist, such as [9], do not end up saving power compared to acquisition of the entire image when dealing with sensors, as in our case, for the visible light spectrum.

1.4 Outline of the Thesis

In Chapter 2 we present a method for acquiring image corners from folds and show simulation performance results of the algorithm on real images. In Chapter 3 we present algorithms for determining the translation and rotation between two subsequent image frames from folds and show simulation results. In Chapter 4 we conclude with some discussion of the findings of this thesis and remarks.

Chapter 2

Compressive Corner Detection

In this section we present an algorithm for extracting corners from folded representations of an image. A potential application of this algorithm is vision-based navigation systems where features are tracked from consecutive video frames for motion and navigation estimation. We will first discuss corner detection in general, followed by a description of the algorithm for detecting corners from folded images, and then show simulation results of the algorithm using real images.

2.1 Corner Detection

In order to understand how to compressively extract corners, we must first strictly define what we mean by a 'corner'. To a human it seems rather simple: there is a corner wherever two edges intersect at close to a right angle. However, as is typically the case in machine vision, we need a more systematic and mathematical approach to extract a corner computationally. Harris and Stephens proposed an approach in 1988 [13] that has become a widespread standard for the task. Intuitively, it strives to find pixels that have drastic changes in intensity in two orthogonal directions. One can think of this as there exists a corner wherever the sum of squared differences between an image patch and the shifted version of the patch by 1 pixel in the horizontal and vertical directions has a large value. If a large difference is observed only when shifting in the horizontal or vertical direction, then the image patch contains an

edge. If no large response is observed when shifting in any direction, then the image patch is of relatively constant intensity. Harris extended this simple method of corner detection to be isotropic by incorporating all possible shifts, not only in the horizontal and vertical direction, by approximating the sum of squared differences via a Taylor expansion. Following is a formal statement of the algorithm:

Let a grayscale image I be the input the algorithm. $I[x_1, x_2]$ represents the image intensity at pixel location $[x_1, x_2]$.

Harris Corner Detection

1. *Calculate discrete approximations to the image gradient in both the horizontal and vertical directions,* $I_x[x_1, x_2]$ *and* $I_y[x_1, x_2]$*, calculated as* [1]

$$I_x[x_1, x_2] = I[x_1, x_2] * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$I_y[x_1, x_2] = I[x_1, x_2] * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

2. *Form a circular smoothing window,* $w[x_1, x_2]$*, typically derived from a two dimensional Gaussian,* $e^{-\frac{x_1^2+x_2^2}{2\sigma^2}}$*, of standard deviation* σ*.*

3. *Create the Harris matrix* M *at every pixel location,* $[x_1, x_2]$*, defined as:*

$$M[x_1, x_2] = \begin{bmatrix} (I_x^2 * w)[x_1, x_2] & (I_x I_y * w)[x_1, x_2] \\ (I_x I_y * w)[x_1, x_2] & (I_y^2 * w)[x_1, x_2] \end{bmatrix}$$

4. *Calculate the corner response function* $R(x, y)$ *defined as:*

$$R[x_1, x_2] = determinant(M[x_1, x_2]) - k * trace(M[x_1, x_2])$$

[1] The symbol $*$ indicates 2-dimensional convolution. Two important detail of its implementation in this thesis are: 1) The output image size is truncated to be equal to the larger of the two input image sizes and 2) edge values are replicated for the computation of output values where the convolutional mask extends past the input image boundaries; this reduces edge effects from the convolution.

If we let α and β be the eigenvalues of the matrix $M[x_1, x_2]$, then corner pixels are located where both α and β are large. It can be computationally expensive to calculate the eigenvalues at all pixel locations, so an approximation to this explicit calculation is achieved with the above corner response function $R[x_1, x_2]$. In this expression, k is a sensitivity factor. Smaller values of k make the algorithm more likely to detect sharper corners. ■

The Harris algorithm outputs the corner response function $R[x_1, x_2]$. It is a distribution over all the pixels of corner likelihoods. Larger values of R indicate a stronger corner response for that pixel and thus an increased likelihood of being a corner.

The next step in most corner detection applications is to extract exact corner locations from the response function $R[x_1, x_2]$. This traditionally involves two steps:

- **Non-maximal Suppression**. This step involves "thinning" the response matrix by shrinking neighboring connected large response values (corresponding to the same corner) to the single local maxima. This avoids outputting two adjacent pixels as separate corners when in reality they have large corner response values $R[x_1, x_2]$ due to the same corner region in the image.

- **Corner Identification**. This step involves picking a subset of the largest corner response values (after non-maximal suppression) as the output of the algorithm. The desired method for doing so is largely application specific, but two common constraints are returning the N best corners (N largest corner response values $R[x_1, x_2]$) or returning all corners with a response function value above a user-specified quality level Q.

The implementation of Harris corner detection used in this thesis is the MATLAB `corner` function. The window function w is a 5 x 5 two-dimensional Gaussian kernel

with standard deviation $\sigma = 1.5$:

$$w = \begin{bmatrix} 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0351 & 0.0683 & 0.0853 & 0.0683 & 0.0351 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \end{bmatrix}$$

A sensitivity factor $k = 0.04$ is used. Additionally, a maximum of $N = 200$ corners is returned per image and only corners with a response function value above quality level Q equal to 1% of the maximum response value is returned, namely only those pixels with $R[x_1, x_2] > .01 * max(R[x_1, x_2])$.

2.2 The Algorithm

The input to the algorithm are two distinct folds, $F_1[x_1, x_2] = \texttt{FOLD}(I, p_{1a}, p_{1b})$ and $F_2[x_1, x_2] = \texttt{FOLD}(I, p_{2a}, p_{2b})$, where the fold sizes of each dimension are pairwise coprime, that is $gcd(p_{1a}, p_{2a}) = 1$, $gcd(p_{1b}, p_{2b}) = 1$. The significance of this will be explained in step 3 of the algorithm. The output of the algorithm are $[x_1, x_2]$ pixel locations that are candidate corner locations. The algorithm consists of three distinct steps:

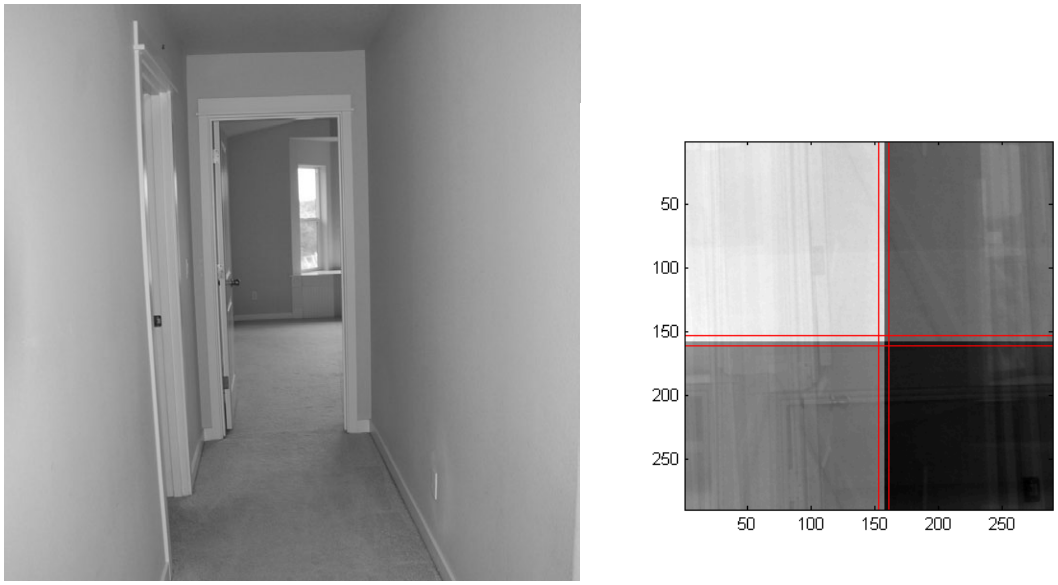
1. Corner Detection
2. Pairwise Correlation
3. Matching & Decoding

These steps will be described in detail below.

1. Corner Detection

In this step a modified version of Harris Corner Detection, as described in section 2.1, is carried out on the two folds, $F_1[x_1, x_2]$ and $F_2[x_1, x_2]$. The departure from standard

Harris detection is that a certain subset of the corner response values is ignored when selecting the best corners. This is to deal with the fact that the folding process creates a prominent false edge in the resultant image in both the horizontal and vertical directions. This occurs when the fold size does not perfectly divide the image dimension, which in general usually occurs. False edges are created when the final subimages in the folding at the extremes of the image (the right and the bottom) add value to only a subset of the folded image pixels. This phenomenon creates a sharp intensity drop off in the form of an edge in the horizontal and vertical edges. This artificial edge is problematic for corner detection when many false corners are created from other natural edges in the image folding on top of this false edge. Figure 2-1 illustrates this phenomenon. If the input image I is of size M x N then the artificial horizontal and vertical edges occur at $M \pmod{p_{ia}}$ and $N \pmod{p_{ib}}$ where i is the index of the current fold.



(a) Original (b) Fold

Figure 2-1: The artificial edges created by folding and the corresponding regions of the corner response function $R[x_1, x_2]$ that are ignored.

It is known a priori that false corners will deterministically be created along these artificial edges from the folding. In order to compensate, all corner response values $R[x_1, x_2]$ within a distance of d pixels from the artificial edge locations, are ignored

in selecting the best corners to output. Figure 2-1 shows the ignored regions for an example fold. This has the the side affect of throwing out any original corners that happen to fold into the ignore regions. Assuming a somewhat uniform distribution of corners in the input image, this does not result in the loss of too many original corners. Specifically the probability of loss from landing in the ignore regions for any given corner is

$$P(loss) = \frac{(2d+1)M + (2d+1)N - (2d+1)^2}{MN}$$

The output of this step of the algorithm are two sets of detected $[x_1, x_2]$ corner locations, c_1 and c_2, from the two input folds F_1 and F_2.

2. Pairwise Correlation

In this step the normalized cross correlation[16] between image patches centered on the identified corners from the previous step is calculated. Square image patches of size $2b+1$ by $2b+1$ are used, where b is the distance from the center pixel to the edge of the image patch. Normalized cross correlation is used instead of standard cross-correlation for a couple of reasons. First, the output values are normalized to the range $[-1, 1]$ which is a desirable property for relative comparisons between image patches. Second, it is more robust to lighting and amplitude changes between images, which we expect to be true from the degradation of folding. The *signal noise* from the folding will yield image patches corresponding to the same corner in the original image that are unequal in the two folded versions. However, in this step we are leveraging the fact that enough image content is preserved in the local neighborhood for template matching between patches to enable corner matching in the next step.

We are specifically only interested in the cross-correlation value at a shift size of zero. In other words, we are taking the cross-correlation value of when the image patches are perfectly aligned. If $p_1[x_1, x_2]$ and $p_2[x_1, x_2]$ are the image patches of size $2b+1$ by $2b+1$ centered on the corners, we are comparing the correlation value

defined as:

$$\frac{\sum_{x_1,x_2}(p_1[x_1,x_2]-\bar{p_1})(p_2[x_1,x_2]-\bar{p_2})}{\sqrt{\sum_{x_1,x_2}(p_1[x_1,x_2]-\bar{p_1})^2\sum_{x_1,x_2}(p_2[x_1,x_2]-\bar{p_2})^2}}$$

When a corner lies within b pixels from the edge of an image, the image patch we use must wrap around the other side of the image. This follows naturally from the fact that bin selection under folding is linear modulo the fold size. Accordingly, the wrapped around pixels are the bins that the image pixels from a particular corner's local neighborhood mapped to in the original image.

3. Matching & Decoding

In this step a matching is obtained between the sets of corners c_1 and c_2. The goal is to decode corner location information about the original input image I from the position of the corner in the folded versions of the image. The Chinese Remainder Theorem (CRT) coupled with the fact that the fold sizes, p_1 and p_2, are coprime, are the critical properties that allow the decoding to take place. The CRT is a theorem about the uniqueness of an integer given its residuals among a series of coprime integers. Formally, it is defined as follows [6, p. 874].

Chinese Remainder Theorem. *Let n_1 ...,n_k be integers greater than 1 and pairwise relatively prime. Let $n = n_1 \cdot n_2 \cdot n_3 \cdots n_k$. Then for any integers a_1,a_2,...,a_k, the set of simultaneous equations*

$$x \equiv a_i \pmod{n_i}$$

for $1 \leq i \leq k$ has a unique solution modulo n for the unknown x. ■

The CRT proves the uniqueness for a given x satisfying the given residuals modulo the a_i. An efficient algorithm for finding x is described in [6, p. 874]. We will call this algorithm `CRT-decode` for the remainder of the paper. Its inputs are the coprime bases n_i and the residuals,a_i.

CRT-decode([a_1 **...** a_k],[n_1 **...** n_k]).

1. *Calculate* $n = n_1 \cdot n_2 \cdot n_3 \cdots n_k$.

2. *Calculate* $m_i = n/n_i$, *for* $1 \leq i \leq k$

3. *Calculate* $e_i = m_i^{-1} \pmod{n_i}$, *for* $1 \leq i \leq k$. *That is, calculate the multiplicative inverse of each* m_i *modulo* n_i. *It is guaranteed to exist since* m_i *and* n_i *are relatively prime. Its computation is accomplished via the Extended Euclidean algorithm* [6, p. 860].

4. *Calculate* $c_i = m_i e_i$ *for* $1 \leq i \leq k$.

5. *Return* $x = (a_1 c_1 + a_2 c_2 + ...a_k c_k) \pmod{n}$ *as the decoded solution to the set of simultaneous equations* $x \equiv a_i \pmod{n_i}$■

Once the same corner from the original image has been identified in each of the two folds, we have the residual of its x_1 and x_2 locations among two sets of coprime integers, (p_{1a},p_{2a}) and (p_{1b},p_{2b}). Therefore running the CRT decoding algorithm along the first dimension and then again along the second dimension yields the location of the corner in the original input image.

The remaining challenge is to match corners between the folds. This is framed as a minimum cost bipartite graph matching problem, to which there are many known solutions. The two sets of corners from each of the folds, c_1 and c_2, form the two disjoint sets of the bipartite graph. An edge is placed between each node of one set to all other nodes in the other set, with a weight of $1-\alpha$, where α is the correlation value between the two corner patches from the previous step. $1 - \alpha$ is used instead of α in order to transform our problem into a minimum cost matching. Next, two important pruning steps take place that reduce the complexity of the graph and ultimately the number of false corners returned.

- **Weak Correlation Values**. We can assume that, even with the noise induced from folding, very weakly correlated image patches do not correspond to the same corner. Accordingly, the graph edges for all corner pairs with a correlation value below a minimum threshold, τ, are removed. Correspondingly all graph edges with a weight greater than $1 - \tau$ are removed.

- **Impossible Corner Matches**. In our case of two folds, the Chinese Remainder Theorem provides a mapping for each dimension from the two residuals modulo the fold sizes to the set of integers from 0 to the product of the fold sizes. That is, in the first dimension a mapping from $[r_1, r_2] \rightarrow 0 \ldots p_{1a}p_{2a} - 1$ where $0 \leq r_1 < p_{1a}, 0 \leq r_2 < p_{2a}$, and for the second dimension, $[r_1, r_2] \rightarrow 0 \ldots p_{1b}p_{2b} - 1$ where $0 \leq r_1 < p_{1b}, 0 \leq r_2 < p_{2b}$. Assuming an input image I of size M by N, the minimum requirements for the system to be able to decode all corner locations according to the CRT is $M < p_{1a} \cdot p_{2a}$ and $N < p_{1b} \cdot p_{2b}$. However, at the feasible levels of compression for decent performance from the algorithm, M and N tend to be significantly less than the product of the fold sizes for their corresponding dimensions. It is this fact that we can exploit in the pruning. Any edge in our bipartite graph that connects two corners who decode to a coordinate outside the original image dimensions can be discarded. We can quantify a precise performance gain from this fact based on the fold sizes. Assuming a uniform distribution of corner locations, the probability of two corners that do not correspond to the same corner in the original image decoding to a location still within the image boundaries is:

$$P(\text{false corner surviving this pruning step}) = \frac{MN}{p_{1a}p_{2a}p_{1b}p_{2b}}$$

  It can be observed that as the fold sizes become larger, this optimization becomes more effective at pruning the complexity of the matching problem.

The specific algorithm used for the matching in this thesis is the classic Hungarian Algorithm presented originally in [15]. The authors in [4] describe an efficient extension for the rectangular case where the number of nodes in each disjoint set of the bipartite graph is different. This is of interest for our application given the number of corners recovered from each fold is in general not equal.

2.3 A Simple Example

We will now walk through the steps of the algorithm on a simple contrived image consisting of only 4 corners. The image is 1024 by 1024 and is shown in Figure 2-2. For simplicity the horizontal and vertical fold sizes for each fold were the same, $p_{1a} = p_{1b} = 157$, $p_{1a} = p_{1b} = 161$.

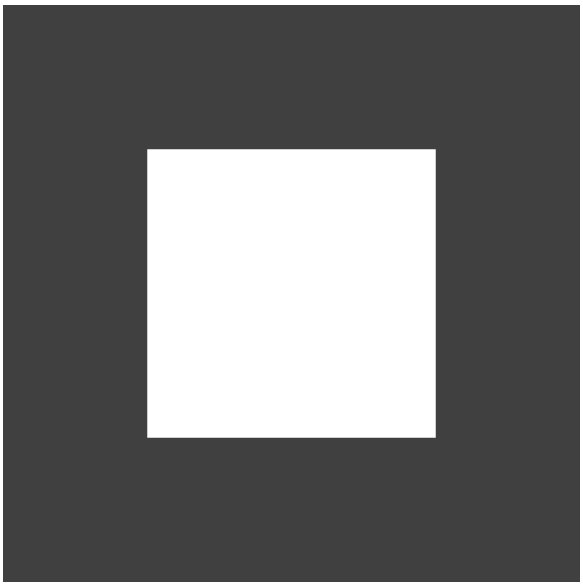
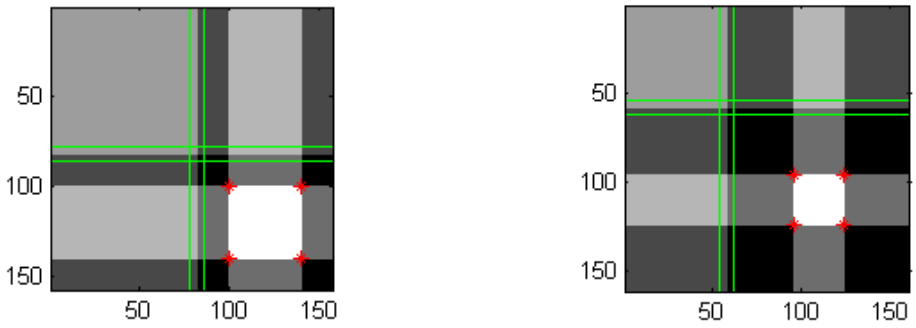


Figure 2-2: The starting image for our algorithm walk-through. It contains 4 distinct corners.

Figure 2-3 shows the output of the first step of the algorithm, corner detection. Figure 2-3a shows the first fold and Figure 2-3b shows the second fold. The red stars indicate the detected corners and the area between the green lines show the corner response values that are ignored.

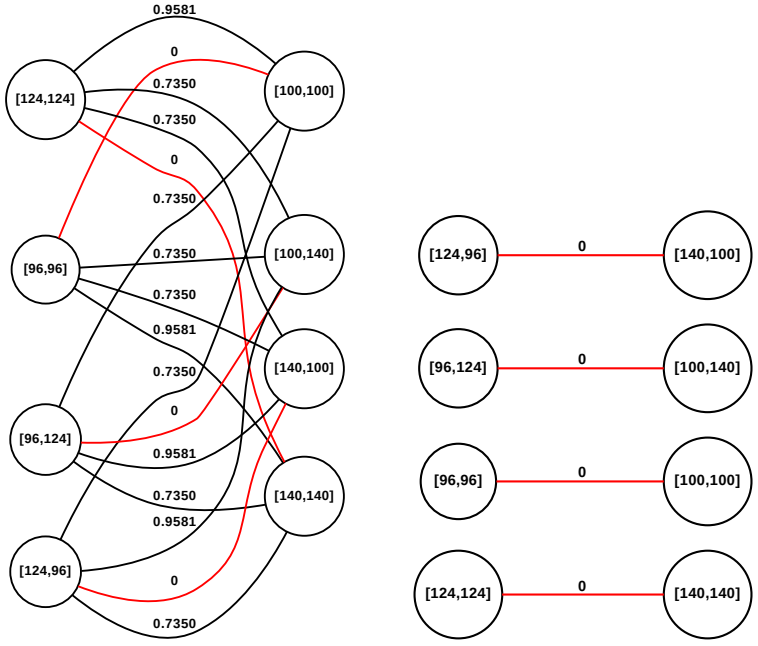
Figure 2-4 shows the second and third steps of the algorithm. The graph prior to the two pruning steps is shown in Figure 2-4a. The reduction in complexity in the bipartite graph from the pruning is shown in Figure 2-4b. In this very simple example, the pruning has done all the work and only one possible matching exists. This is not generally the case where, after the pruning, the Hungarian algorithm is used to determine the final corner correspondences.

Figure 2-5 shows the final output of the algorithm. The decoded corners from the matching using the CRT are shown with red stars. In this case all of the original



(a) Fold 1: $p_{1a} = p_{1b} = 157$ (b) Fold 2: $p_{1a} = p_{1b} = 161$

Figure 2-3: The output of step 1 of the algorithm of corner detection on each subimage. The red stars indicate detected corners. In between the green lines indicate the ignored corner response regions.



(a) Pre-pruning (b) Post-pruning

Figure 2-4: The bipartite matching graphs prior to pruning and after pruning.

corners were correctly recovered and the algorithm completed successfully.

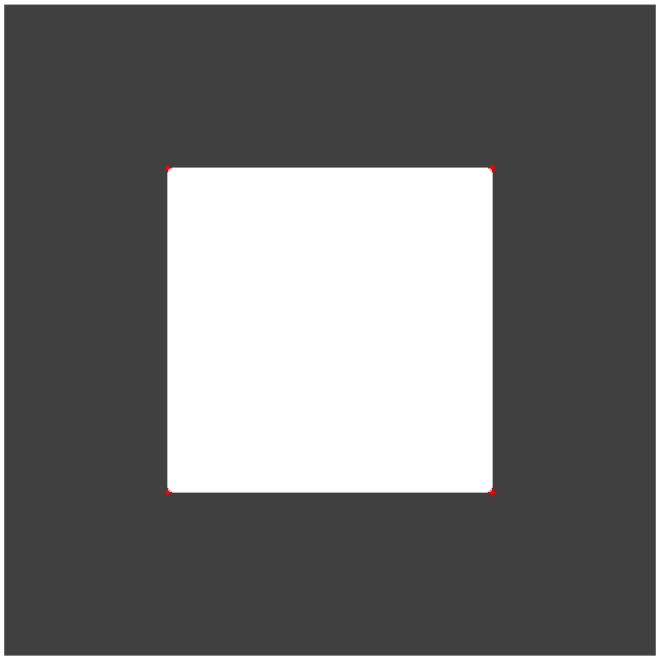


Figure 2-5: The final output of corner detection algorithm. The 4 corners were successfully detected and are indicated with red stars.

2.4 Simulation Results on Real Images

In this section a number of different results will be presented on the performance of the algorithm on real images. In all cases, for simplicity $p_{1a} = p_{1b}$ and $p_{2a} = p_{2b}$. All of the experiments are conducted at a range of compression ratios, showing how the algorithm performs as a function of the degree to which dimensionality is reduced. Compression ratio, for the purposes of this experiment, was defined as the total number of output measurements from the folds divided by the number of pixels in the original picture. All input pictures for the following experiments are of size 1024 by 1024. The specific fold sizes used and corresponding compression ratios are shown in Table 2.1.

An test suite of images was used for the algorithm simulations. They can be seen

in Figure 2-6 [2] [3].

Table 2.1: Fold sizes and corresponding compression ratios for an input image size of 1024 by 1024.

$p_{1a} = p_{1b}$	$p_{2a} = p_{2b}$	Compression Ratio
101	104	0.0200
143	147	0.0401
176	179	0.0601
203	207	0.0802
227	231	0.1000
249	253	0.1202
269	273	0.1401
287	291	0.1593
305	309	0.1798
322	325	0.1996
338	341	0.2198
353	357	0.2404
367	371	0.2597
382	385	0.2805
395	399	0.3006
409	412	0.3214
421	424	0.3405
433	436	0.3601
445	448	0.3803
459	457	0.4001

2.4.1 Performance

Figure 2-7 shows the results of our compressive corner detection algorithm on the test suite of images. Figure 2-7a shows the number of correctly recovered corners as a function of the compression level. Figure 2-7b shows the number of false corners returned by the algorithm as a function of the compression level. Performance appears to scale linearly with the level of compression, and the number of false corners tends to stay at approximately a constant level. A corner is defined as correctly recovered if it is within 3 pixels of a corner returned from running corner detection on the original

[2] Image 1 source: retrieved online at `http://tellerallaboutit.files.wordpress.com/2010/06/upstairs-hallway-before.jpg`

[3] Images 2,3, & 4 source: photographed by the author

(a) Image 1

(b) Image 2

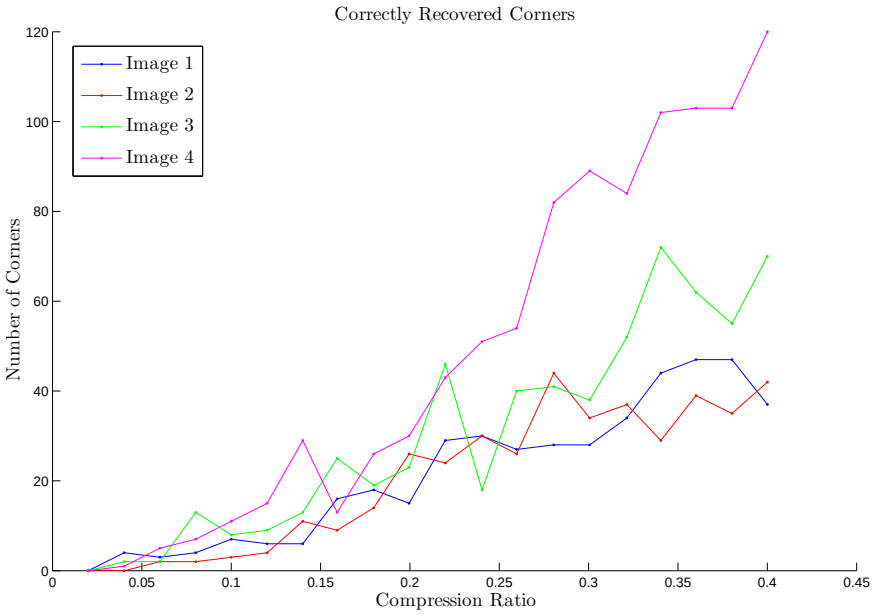
(c) Image 3

(d) Image 4

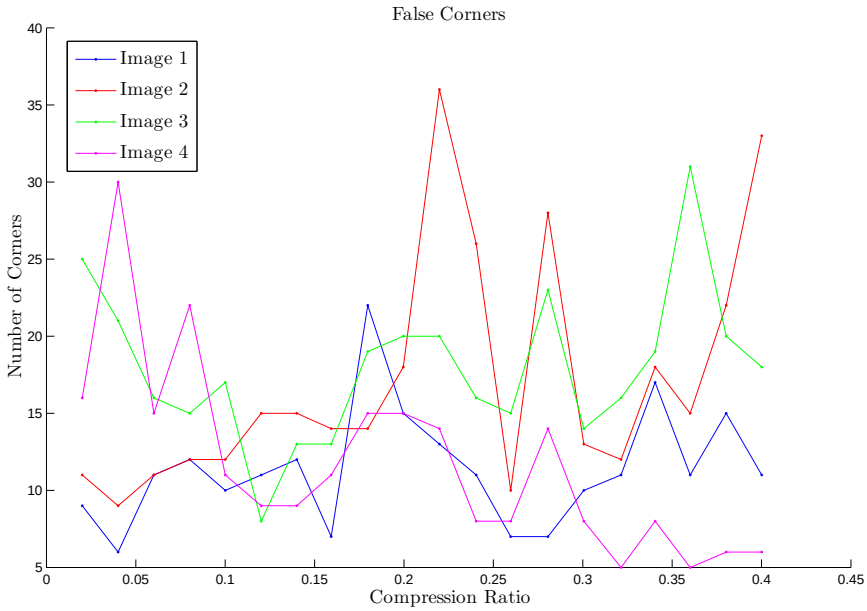
Figure 2-6: The test images used in the simulations.

unfolded version of the image. All other returned corners were classified as false corners. A more useful view of the results for the purposes of comparing performance between the images is shown in Figure 2-8. It is the same graph as Figure 2-7a with each curve normalized by the number of corners detected on the unfolded original version of each image. Thus it shows the percentage of original corners detected across the compression ratios on our test suite. This is a more fair comparison across different images because there is a fundamentally different amount of corners returned

by Harris corner detection for every image.



(a) Correctly Recovered Corners



(b) False Corners

Figure 2-7: The performance of our algorithm on the test suite.

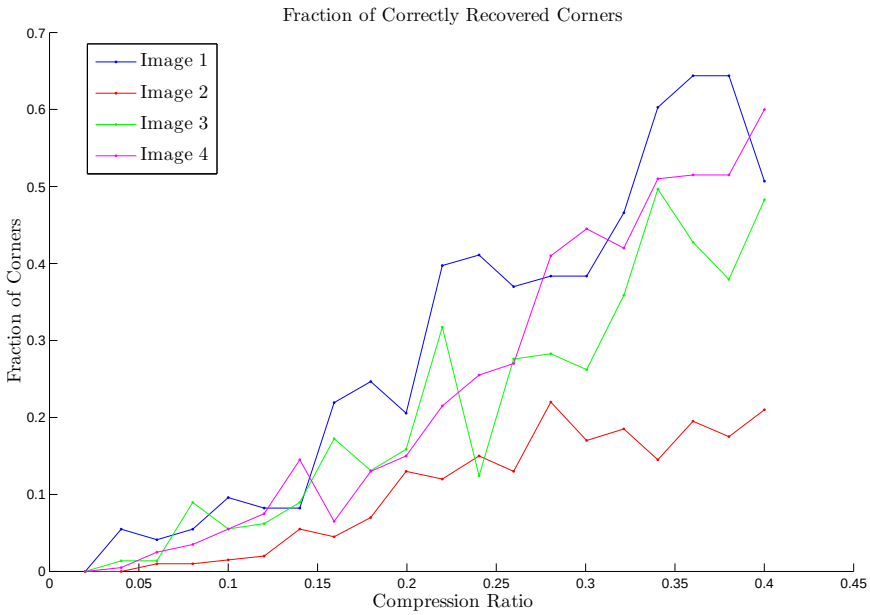


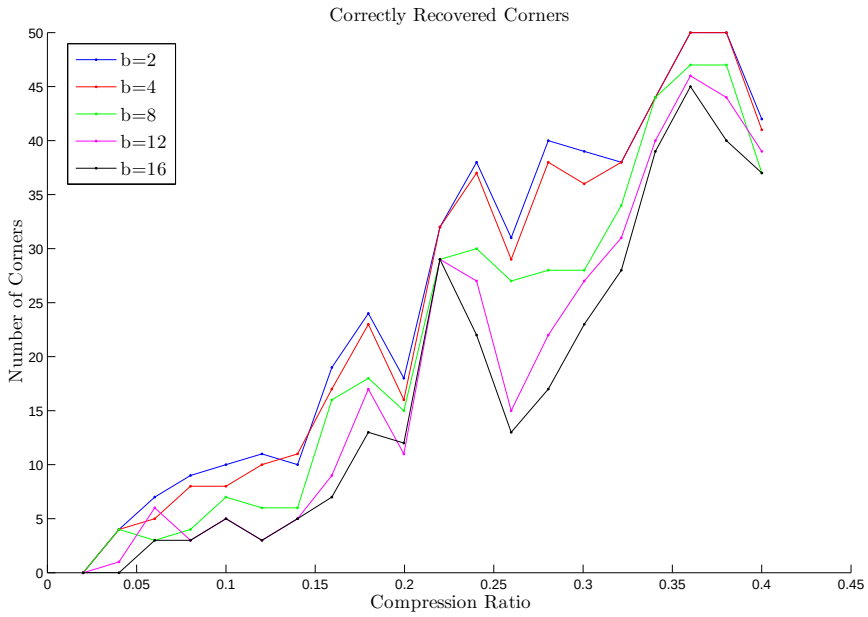
Figure 2-8: The performance of our algorithm on the test wuite showing the percentage of original corners recovered. These curves are the same as in Figure 2-7a, but normalized by the number of corners in the original image.

2.4.2 Parameter Variation

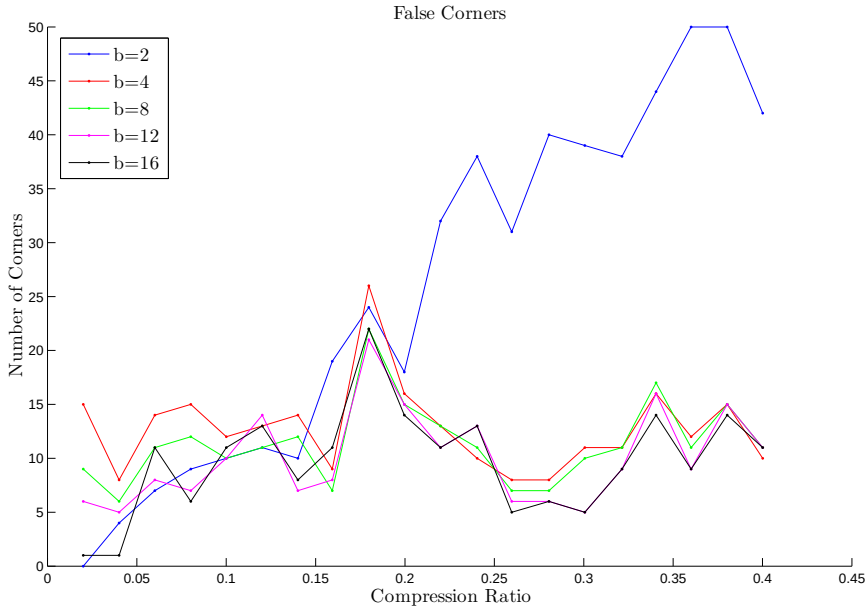
In this section the parameters b and τ are varied to show the performance effects and analyze the trade offs in determining optimal values.

Correlation Window Size, b

Figure 2-9 shows the performance of the algorithm for different values of b on Image 1 from the test suite in Figure 2-6a. We observe that when the window size is very small, $b = 2$, the surrounding image patch of a corner is too small to allow differentiation between corners during the pairwise correlation. At the other extreme, when it is too large, correlation values remain too small because of the added noise from the folding process. With $b = 8$ there appears to be a balanced trade off between the two competing effects, however the optimal value for this parameter would ultimately be application specific depending on the desired performance metric.



(a) Correctly Recovered Corners - Different Values of b



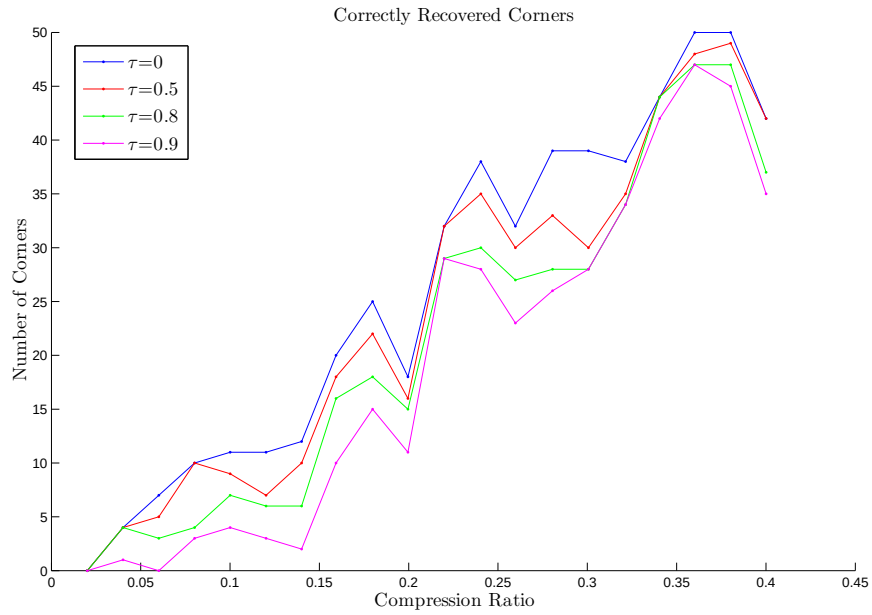
(b) False Corners - Different Values of b

Figure 2-9: The performance of our algorithm on Image 1, 2-6a, for different values of b.

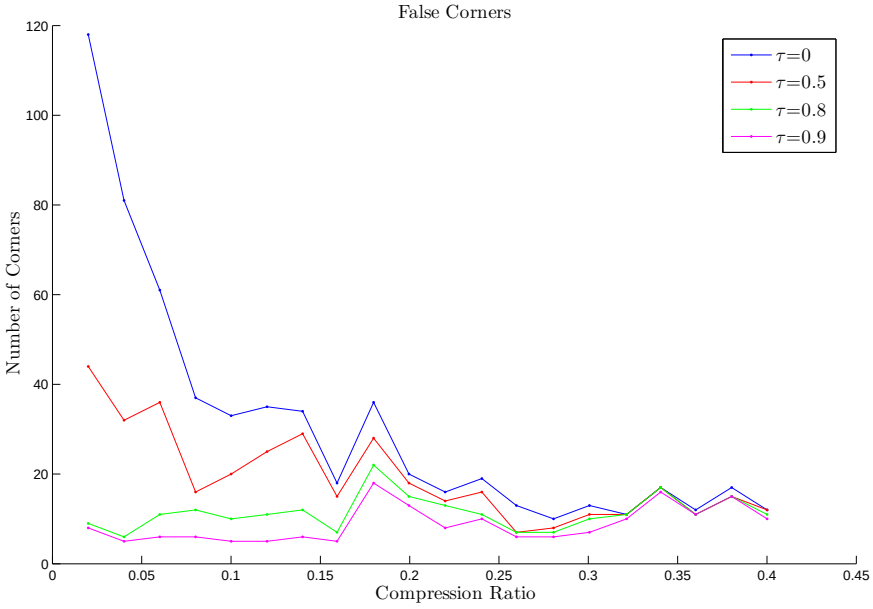
Correlation Value Threshold, τ

Figure 2-10 shows the performance of the algorithm for different values of τ on Image 1 from the test suite in Figure 2-6a. A value of zero for τ corresponds to no pruning

based on correlation value and increasing τ corresponds to relaxing the minimal standard for a correlation value to be considered a match. From Figure 2-10a we can see that increasing τ causes a slight decrease in the number of correctly recovered corners. From Figure 2-10b, it can be seen that increasing τ causes the number of false corners to decrease significantly, especially at the lower compression levels where the pruning from impossible corner matches is less effective. As is the case for the parameter b, the optimal value for τ depends on the application and what performance metric the user wishes to maximize. For the experiments with the test suite in Figure 2-6, a value of $\tau = 0.2$ is found to be a good trade off between the two competing effects.



(a) Correctly Recovered Corners - Different Values of τ



(b) False Corners - Different Values of τ

Figure 2-10: The performance of our algorithm on Image 1, 2-6a, for different values of τ.

2.4.3 Repeatability

From the results above, it can be seen that the algorithm detects a subset of the original corners in the image. In order for the algorithm to be applicable to vision-based navigation, we must verify that a completely different subset of corners is not returned from subsequent frames of the same scene. This would yield frame-to-frame feature tracking impossible. Features must survive temporally across subsequent image frames in order for the vision-based navigation algorithms to succeed. Figure 2-11 shows consecutive image frames of two different scenes. Figure 2-11b is a translated version of Figure 2-11a and Figure 2-11d is a rotated version of Figure 2-11c. The red dots indicate the correctly decoded corners from each image. A common set of 22 corners were detected between Figures 2-11a and 2-11b, and a common set of 28 corners were detected between Figures 2-11c and 2-11d. While these two simple tests are not conclusive, they suggest the algorithm can provide a reasonable set of stable corners across aspect changes of the same scene. This is an aspect that needs to be verified in greater detail in future work.

2.5 Discussion

As the results above show, at very high compression levels virtually all corners are lost. This is the result of multiple compounding effects.

- Higher compression levels correspond to smaller folds and consequently increased degradation from the folding process. Smaller folds correspond to a larger number of pixels from the original image mapping to any given bin of the folded image. The local geometry of many corners are simply lost in the folded versions with so many image patches being summed on top of each other. This manifests itself at two different places within the algorithm. First, some corners are simply not detected in the folded versions. Second, some corners are still detected in both folded representations but have such low correlation value from the noise that they cannot be matched.

(a) Image 1

(b) Image 1 Translated

(c) Image 3

(d) Image 3 Rotated

Figure 2-11: The sets of decoded corners from aspect changes to two different scenes.

.

- Smaller fold sizes cause an increased number of fake corners to be detected in the folded versions. Fake corners are two edges from the original image folding on top of each other in the folded version thus forming a corner. Fake corners increase the complexity of the matching by adding more nodes to the bipartite graph matching problem. When a fake corner is accidentally matched by the algorithm, a false corner is returned.

- Smaller fold sizes decrease the effectiveness of both types of pruning during

the matching phase. Weak correlation pruning starts to prune correct matches accidentally because correlation values are so low from the increased folding noise. Impossible corner pruning becomes less effective because the product of the fold sizes is smaller and consequently the probability of a false corner surviving the pruning step increases.

All of the above effects are alleviated as the compression rate decreases (i.e. fold sizes increase) and this can be observed in the performance graphs as can be expected from natural images.

There is always some loss of corners through the folding process and always some amount of false corners returned by the algorithm. From the perspective of vision-based navigation, this may be tolerable since feature-tracking only needs some subset of consistent features from frame to frame. Additionally, features the system cannot track, such as false corners, are usually ignored. More work needs to be done to verify this conjecture and to optimize the algorithm, but nevertheless we are left very hopeful that a folding image architecture has the potential to provide a low power approach for future vision-based navigation applications.

Chapter 3

Compressive Image Rotation and Translation Determination

In this section we present algorithms for extracting rotation and translation information from folded representations of an image. An especially relevant application is aerial photography, such as a series of image or video frames from a satellite or reconnaissance aircraft. The motion model for images in such scenarios is typically constrained to rotation and translation due to the geometry of the situation. A similar application to vision-based navigation systems is a low-cost ground imager on the bottom of a UAV that provides rotation and translation vectors with respect to the Earth's surface to be used in calculating navigation solutions.

In order to understand the algorithms for folded rotation and translation determination, we first provide the background on traditional methods for accomplishing the task operating on the entire original image frames. The ideas for the new algorithms presented in this section rely on them. Then we will present algorithms for folded rotation and translation determination followed by simulation results of the algorithms on real images.

3.1 Traditional Methods

Let $I_0(x, y)$ represent an image at time 0 and $I_t(x, y)$ represent an image at time t. The image is said to be *translated* by an amount (x_0, y_0) if

$$I_t(x, y) = I_0(x - x_0, y - y_0)$$

The image is said to be *rotated* by an angle θ_0 if

$$I_t(x\cos\theta_0 - y\sin\theta_0, x\sin\theta_0 + y\cos\theta_0) = I_0(x, y)$$

The two can be combined such that an image is said to be translated by an amount (x_0, y_0) and rotated by an amount θ_0 if

$$I_t(x\cos\theta_0 - y\sin\theta_0 + x_0, x\sin\theta_0 + y\cos\theta_0 + y_0) = I_0(x, y)$$

Phase Correlation

Kuglin and Hines presented a technique in [14] for determining translation between two images based on the phase difference between the Fourier transforms. Let $I_1[x_1, x_2]$ and $I_2[x_1, x_2]$ be the two input images where I_2 is a translated version of I_1 by an amount $[x_{1t}, x_{2t}]$, and let F_1 and F_2 be the corresponding two-dimensional Discrete Fourier Transforms, $F_1[m_1, m_2] = DFT\{I_1[x_1, x_2]\} = |F_1|\, e^{j\phi_1}$ and $F_2[m_1, m_2] = DFT\{I_2[x_1, x_2]\} = |F_2|\, e^{j\phi_2}$. A phase correlation function $d = DFT^{-1}\{e^{j(\phi_1 - \phi_2)}\}$ is constructed and, if I_2 is a translated version of I_1, the phase correlation function d will have a spike exactly at $[x_{1t}, x_{2t}]$.

This follows from the Fourier Shift theorem, which states that if there are two signals of size M x N, I_1 and I_2, where one is a cyclically shifted version of the other, the discrete Fourier transforms are related by:

$$F_2[m_1, m_2] = F_1[m_1, m_2] \cdot e^{-j\frac{2\pi}{M}m_1x_{1t}} \cdot e^{-j\frac{2\pi}{N}m_2x_{2t}}$$

The phase difference, $\phi_1 - \phi_2$, will be a constant phase offset term and the inverse transform will yield an impulse in the phase correlation function d at the the shift amount. It should be noted that in most applications the subsequent images are not cyclically shifted but rather *infinitely* shifted. That is, only a subset of the image data is common, and new image content replaces the circularly wrapped around portion. Consequently, this equality does not strictly hold but still provides a good approximation, especially at small shift sizes.

From an implementation perspective, the phase difference, $\phi_1 - \phi_2$, can be thought of as the phase of the cross-power spectrum of the two signals:

$$\frac{F_1 F_2^*}{|F_1 F_2^*|}$$

Eliminating the normalizing factor above and inverse transforming is equivalent to taking the cyclic cross-correlation between the two images and looking for a peak. Although this method can also be used to accurately register the images, Kuglin and Hines argue that pure phase correlation is superior for a number of reasons, including robustness to noise and elimination of ambiguities in maximal peak selection.

Rotation Detection

De Castro and Morandi extended the work of Kuglin and Hines to incorporate rotation registration into the framework as well [7]. They recognized that for infinite 2-d signals and continuous transforms, the Fourier Rotation Theorem applies. That is, if $I_t(x, y)$ is a rotated version of a signal $I(x, y)$ by an amount θ_0, then their Fourier Transforms are related by:

$$F_t(\omega_1, \omega_2) = F(\omega_1 \cos\theta_0 + \omega_2 \sin\theta_0, -\omega_1 \sin\theta_0 + \omega_2 \cos\theta_0)$$

The rotation amount, θ_0, is then found based on how much the corresponding transforms have rotated. Common methods for determining this quantity include converting the transforms to a polar coordinate system where the rotation becomes a

translation, or manually rotating the image while sweeping through a range a values and measuring where the strongest correlation peak occurs. As McGuire observed [19], the equality above does not strictly hold for finite signals and discrete transforms, as the periodic replication effect of the DFT does not commute with rotation. There are optimizations and modifications that can deal with this effect, however the above discussion is sufficient for our purposes and provides the necessary background for the rest of this thesis.

When an image is both translated and rotated, the constant phase offset from translation will not affect the rotation property of the Fourier transform magnitudes. Therefore, the approach is to first undo the rotation by determining it from the transform magnitudes, yielding two purely translated images that phase correlation can operate on to determine the translation amount.

3.2 Compressive Rotation and Translation Determination

In this section we present algorithms to determine translation and rotation amounts between two subsequent images from their folded representations. We seek to establish to what extent translation and rotation information are preserved under folding and what is the best way to extract them. The traditional methods discussed above serve as a starting point for this investigation.

3.2.1 Translation

To understand the compressive algorithm for determining translation from folded representations, we first analyze how translation is preserved throughout the folding process. As it turns out, it is preserved very well. In order to see this we must remember the definition of folding used in this thesis, presented in Section 1.1. Folding is in essence a hash function that maps each pixel in the input image to a bin in the folded image and sums all of the pixels mapped to the same bin. The desirable

property of this framework with respect to translation is that within the common region of overlap between two frames, the sets of pixels mapped to the same bin does not change. Moreover, the specific bin that a same set of pixels is mapped to in the translated version, is exactly the translated bin from the first frame modulo the fold size in each dimension. In other words, restricting ourselves to just the common region of overlap, one fold is exactly a circularly shifted version of the other. For the pixels that are not in the regions of overlap, *noise* is added on top of the circularly shifted images; Figure 3-1 illustrates this phenomenon. The red region indicates the area of overlap between the images and the folding process on the translated images can be seen. The two folded images can be modeled as $f_1 = f_1' + \mu_1$ and $f_2 = f_2' + \mu_2$, where f_1' and f_2' are circularly shifted versions of each other. That is,

$$f_2'[x_1, x_2] = f_1'[x_1 - x_{1t} \pmod p, x_2 - x_{2t} \pmod p]$$

where p is the fold size. μ_1 and μ_2 can be modeled as signal noise from the folding process from the non-common image content between the two frames.

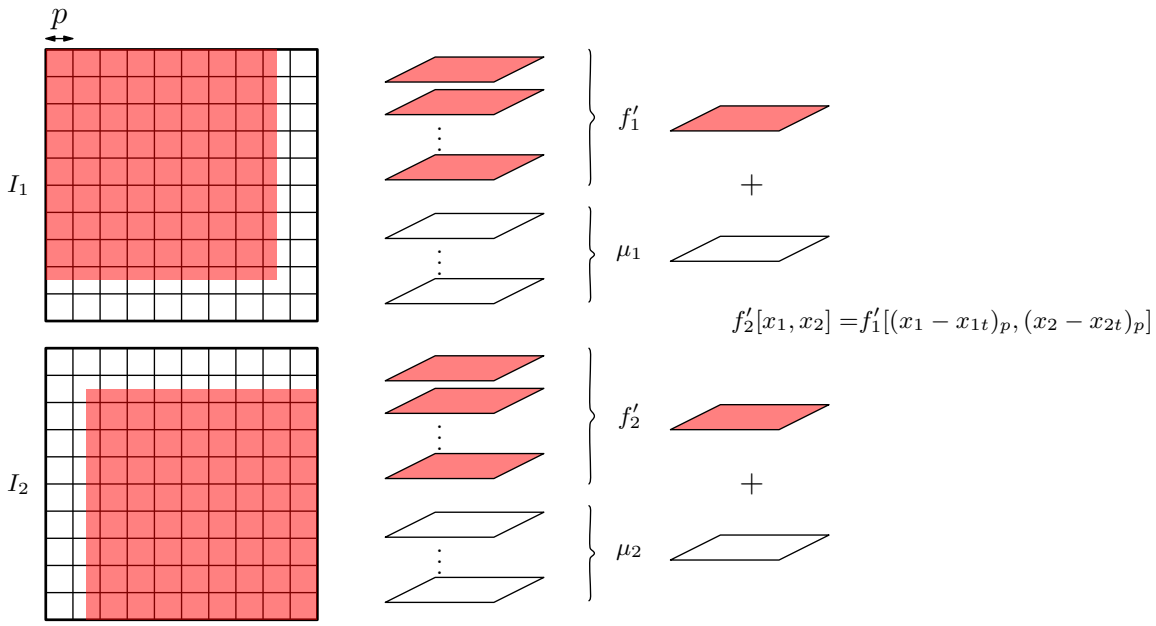


Figure 3-1: The effects of folding on translated image pairs. The red region indicates the overlap. The cyclic shifted subcomponents are f_1' and f_2'.

With this phenomenon in mind, we can now explore the algorithm for determin-

ing the exact translation amount from folded versions. Let $I_1[x_1, x_2]$ and $I_t[x_1, x_2]$ be the input images, where $I_t[x_1, x_2]$ is translated by an amount $[x_{1t}, x_{2t}]$. We will assume our algorithm has access to folded representations of the images. Similar to the corner detection algorithm in Section 2.2, the algorithm operates on two folded versions of both I_1 and I_t, where the fold sizes are coprime in each dimension, that is $gcd(p_{1a}, p_{2a}) = 1$, $gcd(p_{1b}, p_{2b}) = 1$. Accordingly, the inputs to our algorithm will be the two folds of I_1, $F_{11} = \texttt{FOLD}(I_1, p_{1a}, p_{1b})$ and $F_{12} = \texttt{FOLD}(I_1, p_{2a}, p_{2b})$, and the two folds of I_2, $F_{t1} = \texttt{FOLD}(I_t, p_{1a}, p_{1b})$ and $F_{t2} = \texttt{FOLD}(I_t, p_{2a}, p_{2b})$.

Folded Translation Detection Algorithm

1. *Calculate the phase correlation function d_1 between the first folds of each image, F_{11} and F_{t1}, and the phase correlation function d_2 between the second folds, F_{12} and F_{t2}.*

2. *Identify the peaks in the phase correlation functions, d_1 and d_2. Let the coordinate of the peak of d_1 be $[y_1, y_2]$, and the coordinate of the peak of d_2 be $[z_1, z_2]$.*

3. *With the residuals of the translation vector, $[x_{1t}, x_{2t}]$, among two coprime integers in each dimension we recover the translation vector using the Chinese Remainder Theorem: $x_{1t} = \texttt{CRT-decode}([y_1, z_1], [p_{1a}, p_{2a}])$ and $x_{2t} = \texttt{CRT-decode}([y_2, z_2], [p_{1b}, p_{2b}])$.* ■

Figure 3-2 provides a visual illustration of the above steps.

The algorithm leverages the fact that after folding the phase correlation is able to identify and register the translation of a circularly shifted image. There are two parameters that affect the performance of the algorithm: the translation shift amount and the fold size (or the degree of dimensionality reduction). As the translation shift amount increases, the percentage of common image content decreases, and the correlation peaks become increasingly weak and more difficult to distinguish from noisy correlation peaks. Similarly, as the dimensionality gets reduced from smaller fold sizes, performance decreases as more noise slices are added in the μ component

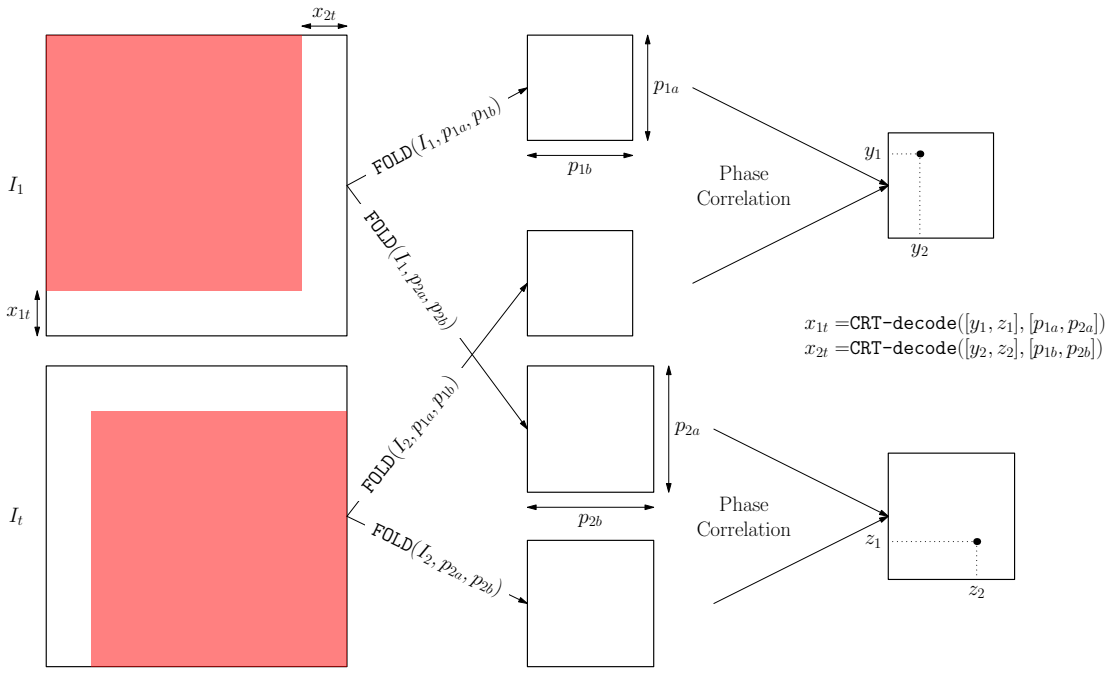


Figure 3-2: An illustration of the folded translation detection algorithm.

of the signal model and the correlation peaks become smaller. In addition, the size of the underlying circularly shifted component becomes smaller with smaller fold sizes and the correlation peaks naturally become weaker because there is less signal to match against.

Complexity

We also would like to note that even without a hardware implementation of folding, our folded translation detection algorithm is still of computational significance. Given input images of size N x N, it has complexity that is linear in the number of image pixels, $O(N^2)$, whereas standard phase correlation operating on the full images has complexity $O(N^2 \log N)$. This is because the algorithm is no longer bound by the transform calculation but by the folding. A derivation of this fact is presented in Appendix A.

3.2.2 Rotation

Unfortunately rotation does not exhibit quite the same circular shifting phenomenon under folding as in the case of translation. In general, after rotation, completely different sets of pixels from the original image are combined in the bins of the folded images. This yields the traditional methods of measuring the rotation amounts of DFT magnitudes useless. However, there is another property of the images that we can utilize: the distribution of gradient directions. Rotation has the effect of rotating every local gradient direction by the same amount. Moreover, strong gradient directions are preserved through the folding process. The local gradient direction is essentially another example of a local geometric feature that tends to be preserved well under folding.

The intuition for the algorithm is that rotation can be determined by examining the distribution of gradient directions between the folded representations of rotated images. The shape of this distribution can be thought of as a signature for the image, and how much that shape shifts by is the rotation amount. The details of the algorithm follow.

The gradient of an image $I(x, y)$ is defined as

$$\bigtriangledown f(x,y) = \frac{\partial f(x,y)}{\partial x}\hat{i}_x + \frac{\partial f(x,y)}{\partial y}\hat{i}_y$$

where $\hat{i}_x$ and $\hat{i}_y$ are unit vectors in the horizontal and vertical direction. In practice we have discrete signals so the gradient is approximated by discrete convolutions. We will use the Sobel operator [17, p. 482] as an approximation here so the partial derivatives for an image $I[x_1, x_2]$ can be approximated as

$$\frac{\partial f(x,y)}{\partial x} \approx I_x[x_1, x_2] = I[x_1, x_2] * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial f(x,y)}{\partial y} \approx I_y[x_1, x_2] = I[x_1, x_2] * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The magnitude of the gradient is defined as:

$$|\bigtriangledown I|\,[x_1, x_2] = \sqrt{I_x^2[x_1, x_2] + I_y^2[x_1, x_2]}$$

The angle, also called direction, of the gradient is defined as:

$$\Theta[x_1, x_2] = \tan^{-1}(\frac{I_y[x_1, x_2]}{I_x[x_1, x_2]})$$

Note that the two-argument arctangent function, $\tan 2^{-1}(I_{x_1}, I_{x_2})$, could also be used if angle range for the $-\pi < \Theta < \pi$ is desired instead of the half resolution,$-\frac{\pi}{2} < \Theta < \frac{\pi}{2}$, provided by $\tan^{-1}$. This change should only be necessary if $|\theta_0| > \pi/2$.

With this in mind, we now present the folded rotation detection algorithm. Let $I_1[x_1, x_2]$ and $I_r[x_1, x_2]$ be the original images where I_r is a rotated version of I_1 by an angle θ_0. The inputs to our algorithm will be the folds of I_1 and I_r, $F_1 =$ $\texttt{FOLD}(I_1, p_a, p_b)$ and $F_r = \texttt{FOLD}(I_r, p_a, p_b)$

Folded Rotation Detection Algorithm

1. *Calculate the gradients of the two input folds (this includes both the horizontal and vertical component for each),* $F_{1_x}, F_{1_y}, F_{r_x}, F_{r_y}$.

2. *Ignore pixels in the regions near the false edges created from the folding process (see Figure 2-1 for a reminder of this phenomenon). From an implementation perspective, do this by setting the gradient to 0 at these pixel values. These pixel values would add very strong arbitrary false votes to angles of* $\Theta = 0$ *and* $\Theta = \pi$ *in the next step so we ignore their values.*

3. *Create a histogram of* N *bins of the gradient angles for each fold. Weight each angle value's vote by the value of the gradient magnitude,*$|\bigtriangledown I|$*, at that coordinate. That is, if* α *is the set of all* $[x_1, x_2]$ *coordinates whose angle* Θ *maps to*

bin i, the histogram value for bin i is $\sum_{x_1,x_2\in\alpha} |\bigtriangledown I|\,[x_1, x_2]$. *Let H_1 and H_r be the N-bin weighted histograms of Θ_1 and Θ_r respectively formed from this step.*

4. *Circularly correlate H_1 and H_r and find the index of the maximum peak. This value, scaled by the number of angles per bin based on the N used for the histograms, yields the rotation amount, θ_0.* ■

Figure 3-3 provides a visual illustration of the above steps.

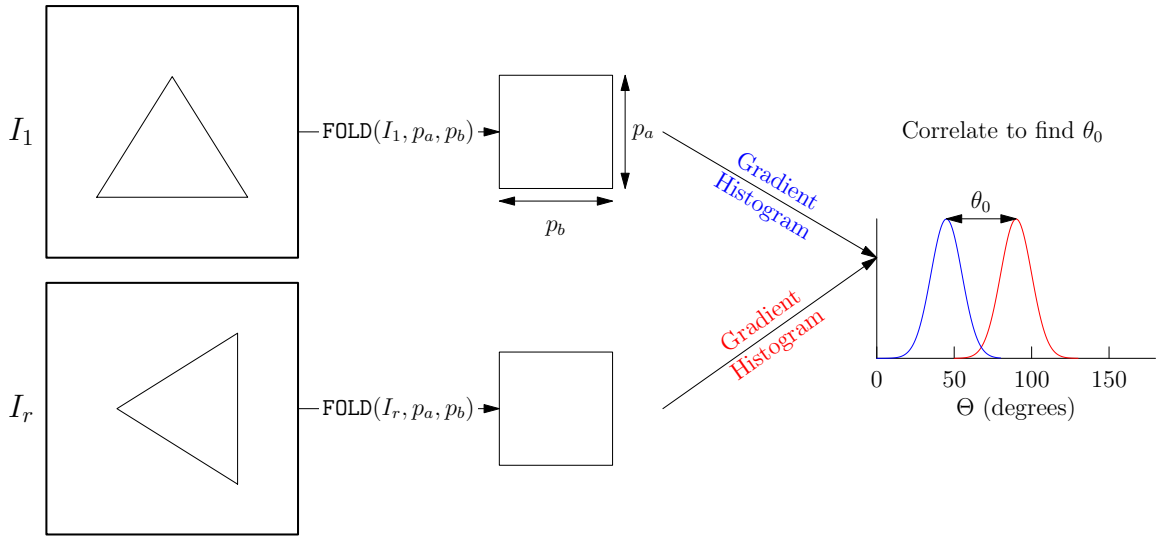


Figure 3-3: An illustration of the folded rotation detection algorithm.

3.3 Simulation Results on Real Images

In this section we present simulation results on real images of the folded translation and rotation detection algorithms presented in the previous section.

Translation Results

For simplicity, in the translation results presented in this section, we consider $p_{1a} = p_{1b}$ and $p_{2a} = p_{2b}$. Figure 3-4 shows the image of size 2048 by 2048 used for the translation experiments [1]. The upper left quadrant is used as the base image, $I_1[x_1, x_2]$, and the

[1]Image source: Retrieved online at `http://www.stockvault.net/photo/download/111396`

red shaded region shows all new image content generated for all possible shifts of $[x_{1t}, x_{2t}]$, for $0 < x_{1t} < 1024$ and $0 < x_{2t} < 1024$. Figure 3-5 shows the results of detecting all of these possible translations at a number of different compression ratios. Red points indicate that the translation amount is correctly identified, and blue that it is not correctly identified. These curves show the boundary where there is not enough common image content anymore to correctly register the images.

As we expected, the boundary pushes outward as the fold sizes increase and the corresponding compression level decreases. However, the results show extremely impressive performance at very high compression levels. Figure 3-5a is very close to the maximum amount of dimensionality reduction our scheme can tolerate ($31*34 = 1054$ which is barely above the necessary 1024 for our image size required to correctly decode the shift given the use of the CRT). Even at this extremely high compression level we can still register an impressive amount of all possible shifts. If we restrict the shift amounts we can tolerate to even smaller sizes than the full extent of the image, this compression level could be pushed down even more so than is shown in the results.

As a point of comparison, Figure 3-5e shows the results of phase correlation on the unfolded image with no compression. The differences between the success boundary in this image and those in Figure 3-5 give a direct measure of the performance loss we give up with folding. The results show that we lose a relatively small amount in performance with a very large amount of dimensionality reduction.

To get a sense for how well our algorithm performs relative to alternative forms of dimensionality reduction, we compare the performance of running our folded translation detection algorithm against traditional phase correlation on reconstructed images using traditional compressive sensing techniques discussed in section 1.3.2. We specifically use noiselets as the measurement matrix, A, and total variation (TV) minimization as the recovery algorithm. It was shown in [20] that this combination tends to work well for natural images. NESTA was the MATLAB optimization package used to carry out the TV minimization [2],[3]. The results are shown in Table 3.1. The original image in all cases is the 1024 by 1024 upper left quadrant of Figure 3-4

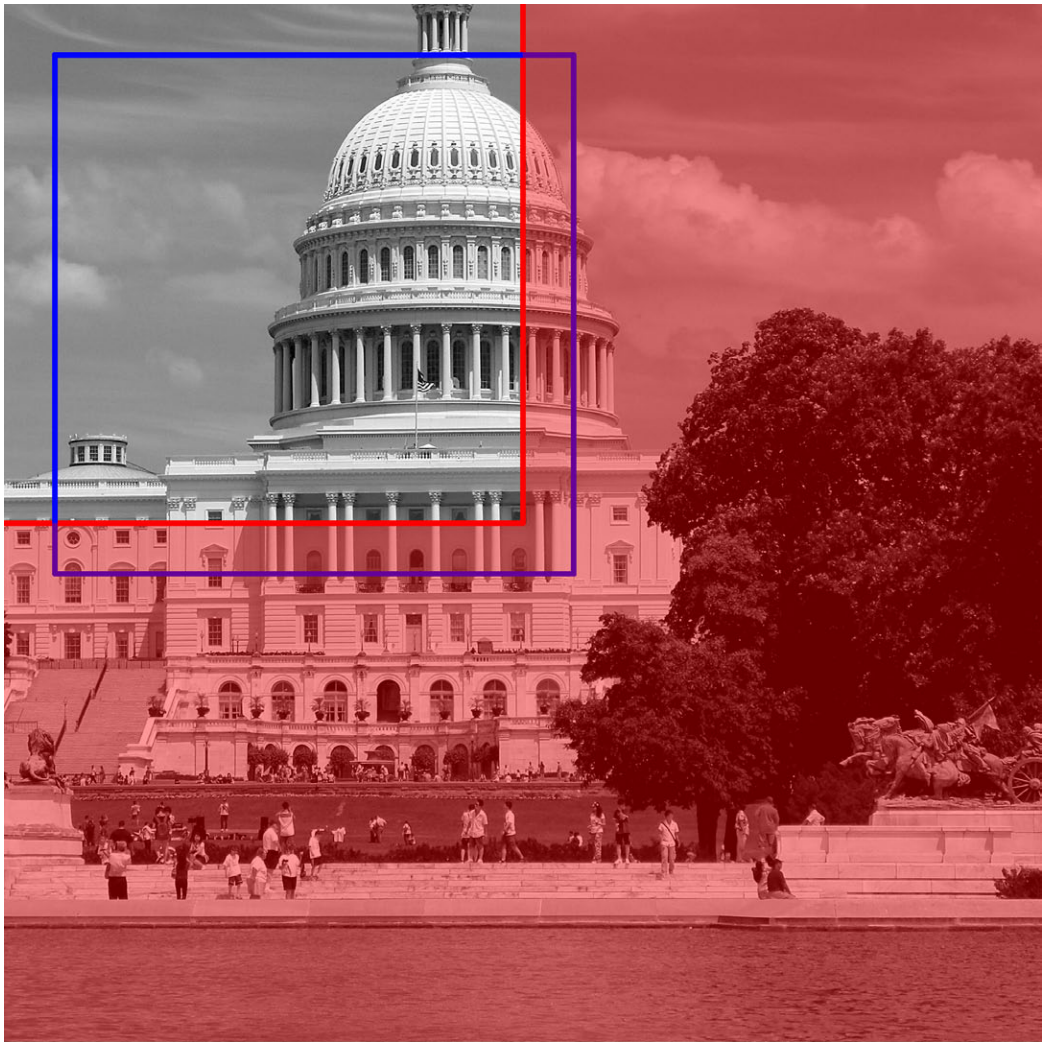


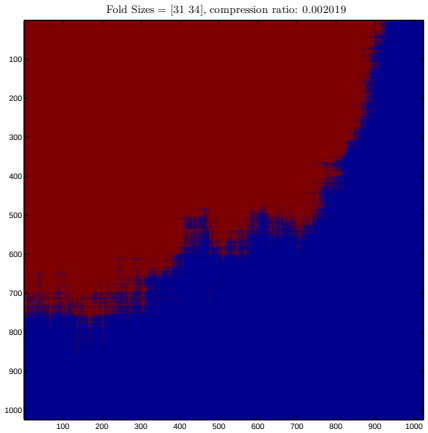
Figure 3-4: The test image used for translation results. The upper left unshaded image was the base image. The red shaded region shows the new image content from all translations tested in the experiments. The blue outlined image shows an example translation of $[100, 100]$.

and a translation of $[100, 100]$ was used, corresponding to the blue outlined image in the same figure. We test performance across a range of compression ratios, and the number of allowed measurements corresponding to the indicated fold sizes and compression ratio is kept the same across the two methods. For instance fold sizes of $[f_1, f_2]$ correspond to $f_1^2 + f_2^2$ total measurements so the M x N measurement matrix A used in traditional compressive sensing method would have $M = f_1^2 + f_2^2$ (and necessarily $N = 1024^2$). Note that since we are restricting the translation amount for this experiment to $[100, 100]$, we can use fold size pairs whose product is less than 1024 but greater than 100 in order to test the algorithms at extremely low compression ratios. However these fold sizes could not be used in general because the translation amount is not known a priori.

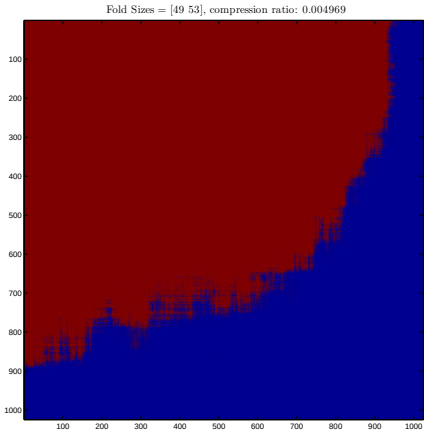
We observe that the folding algorithm significantly outperforms the traditional compressive sensing method. The folding algorithm can accurately register the translation at compression ratios as low as .02%, while the traditional method is ineffecive at any compression ratio lower than about 1.3%. This validates one of the primary motivations of the research: that dimensionality reduction in image feature acquision problems can be furhter increased by using a non-recovery based algorithm that aims to directly acquire the relevant image properties.

Fold Sizes	Compression Ratio	Folding Estimate	Traditional CS Estimate
[11, 12]	0.000253	**[100, 100]**	[31, 182]
[17, 18]	0.000585	**[100, 100]**	[981, 629]
[23, 24]	0.001054	**[100, 100]**	[569, 620]
[33, 34]	0.002141	**[100, 100]**	[845, 131]
[46, 47]	0.004125	**[100, 100]**	[20, 0]
[57, 58]	0.006307	**[100, 100]**	[0, 370]
[65, 66]	0.008183	**[100, 100]**	[0, 98]
[73, 74]	0.010304	**[100, 100]**	[101, 100]
[81, 82]	0.012670	**[100, 100]**	**[100, 100]**
[89, 90]	0.015279	**[100, 100]**	**[100, 100]**
[96, 97]	0.017762	**[100, 100]**	**[100, 100]**
[103, 104]	0.020432	**[100, 100]**	**[100, 100]**
[126, 127]	0.030522	**[100, 100]**	**[100, 100]**
[162, 163]	0.050366	**[100, 100]**	**[100, 100]**
[229, 230]	0.100461	**[100, 100]**	**[100, 100]**

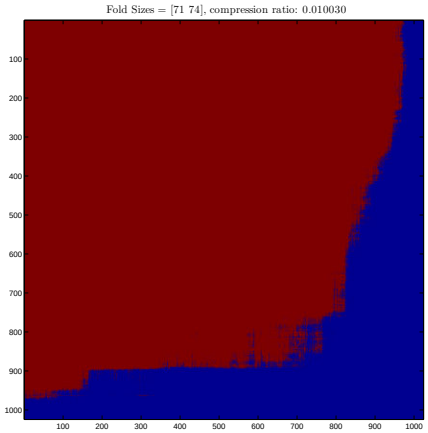
Table 3.1: Results of experiment comparing folding performance to traditional compressive sensing for a translation amount of $[100, 100]$ using the image in Figure 3-4. The bolded red estimates indicate the algorithm correctly registered the image pairs.



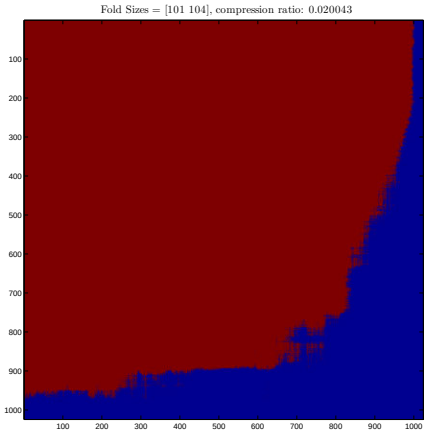
(a) Fold Sizes = (31, 34)



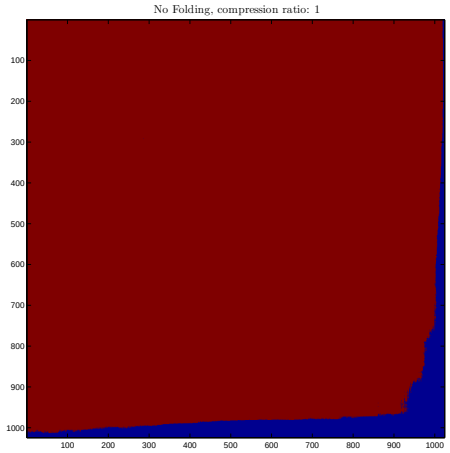
(b) Fold Sizes = (49, 53)



(c) Fold Sizes = (71, 74)



(d) Fold Sizes = (101, 104)



(e) No Folding

Figure 3-5: Results of our compressive translation determination algorithm on the image for the shift amounts shown on the x and y axis. Red regions indicate the translation was correctly detected. Blue regions indicate it was not.

Rotation Results

To test the performance of our folded rotation determination algorithm, we rotate a test image through a series of known rotations and measure the error of our algorithm's output versus ground truth across different compression levels. Figure 3-6 shows the original image used in this experiment[2]. We restricted image content to the shown circle in order to remove the effects of differing image content across the rotations due to occlusion. Because we are rotating by arbitrary angles on a discrete grid, pixels cannot be perfectly moved to their new locations. Bilinear interpolation is used to extrapolate pixel values from a weighting of the four closest neighbors. The MATLAB `imrotate` implementation was used. The rotation amounts are from 1° to 90° at 1° intervals.

Figure 3-6: The test image used on our folded rotation determination algorithm.

Figure 3-7 shows the ground truth rotation and the outputs of our algorithm at various compression levels across the different rotation amounts we perform for the

[2]Image source: photographed by the author

experiment. As expected, we see the algorithm output converging to ground truth as the compression level increases. A more quantitative way of measuring performance is shown in Figure 3-8. The mean squared error of our algorithm vs. the ground truth rotation amount is plotted as a function of dimensionality reduction. The significant aspect of the plots are how performance stabilizes very quickly at low compression rates. From about 3-4% compression all the way to the unfolded version, results are about the same. It should be noted that although the error stabilizes quickly, it does not converge to 0. The exact reason is still unknown and something we are investigating. The error for the unfolded case of no dimensionality reduction can be seen in Figure 3-9. In particular, the algorithm consistently estimates 1° off at rotation amounts near 22.5° and 67.5°. The non-random strucutre of this error and symmetric nature of being centered at angles of $\pi/4 \pm \pi/8$ leads us to believe that this is an artifact of our algorithm and not a fundamental difference between preservation of rotation information through the folding process at different rotation amounts. We believe this is a surrmountable issue.

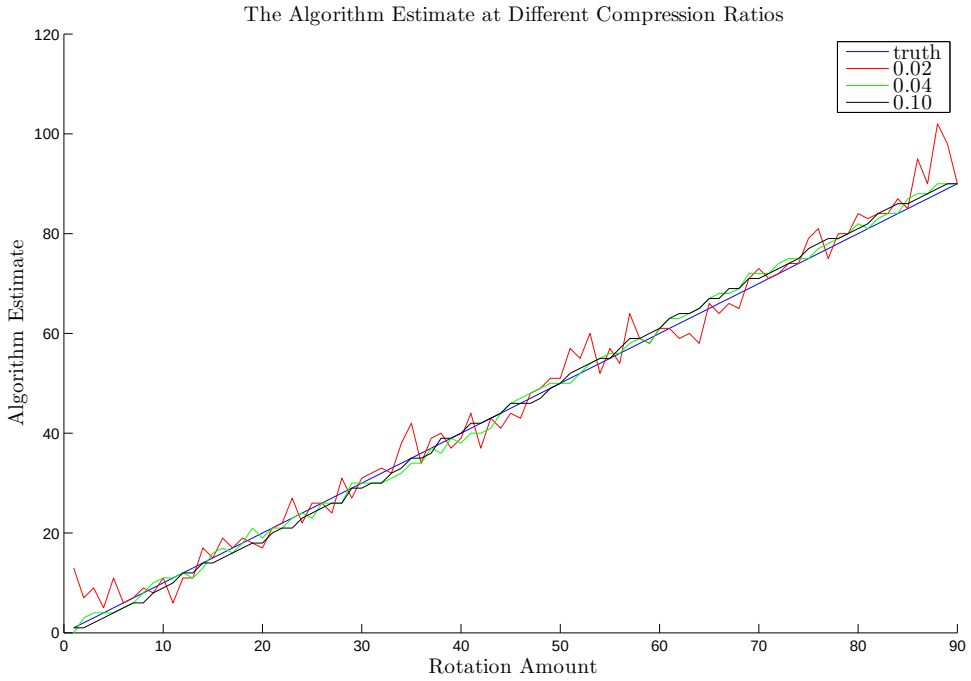


Figure 3-7: The output of our algorithm and ground truth rotation amount for a couple of different compression ratios.

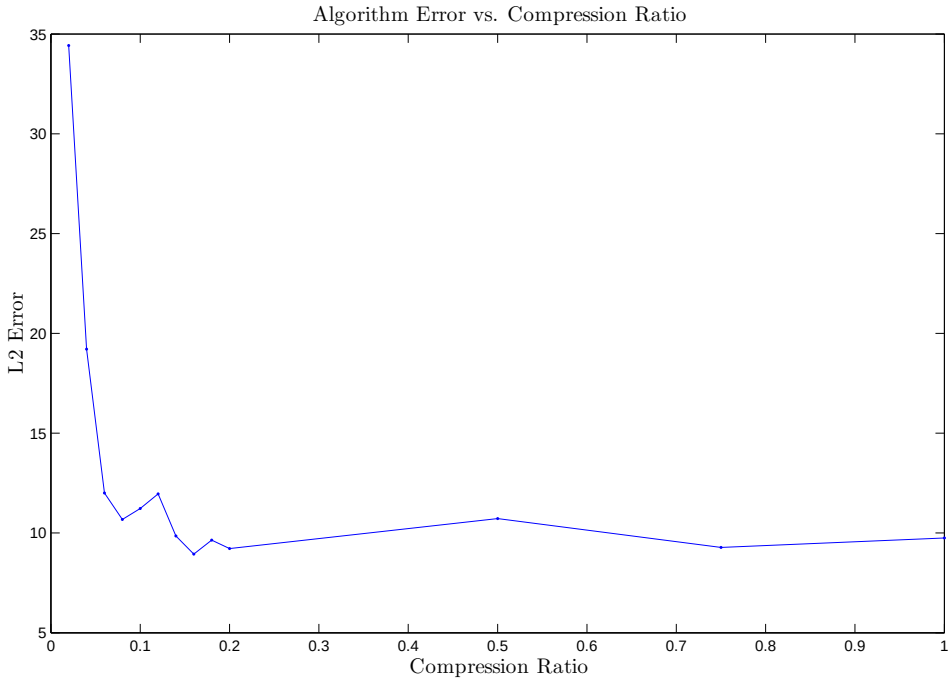


Figure 3-8: The L2 or mean-squared error of the output of our algorithm vs ground truth as a function of compression level.

Another interesting observation is the effect of the histogram size N on the results. We would expect higher histogram sizes to lead to less error because we have higher angle resolution. However, we would also expect there to be an upper bound since histogram mass can be spread out so much that correlation values become too weak to properly discern the rotation amounts. Figure 3-10 shows the mean squared error as a function of compression ratio for a number of different histogram sizes that would be feasible in practice. The results are very similar across different N.

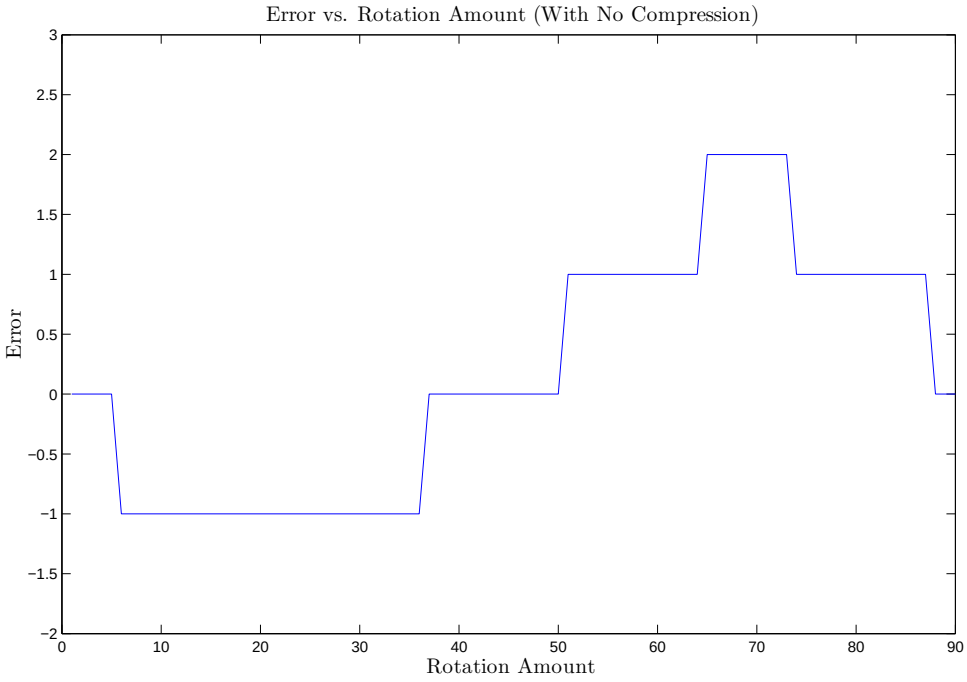


Figure 3-9: The error with no folding as a function of rotation angle.

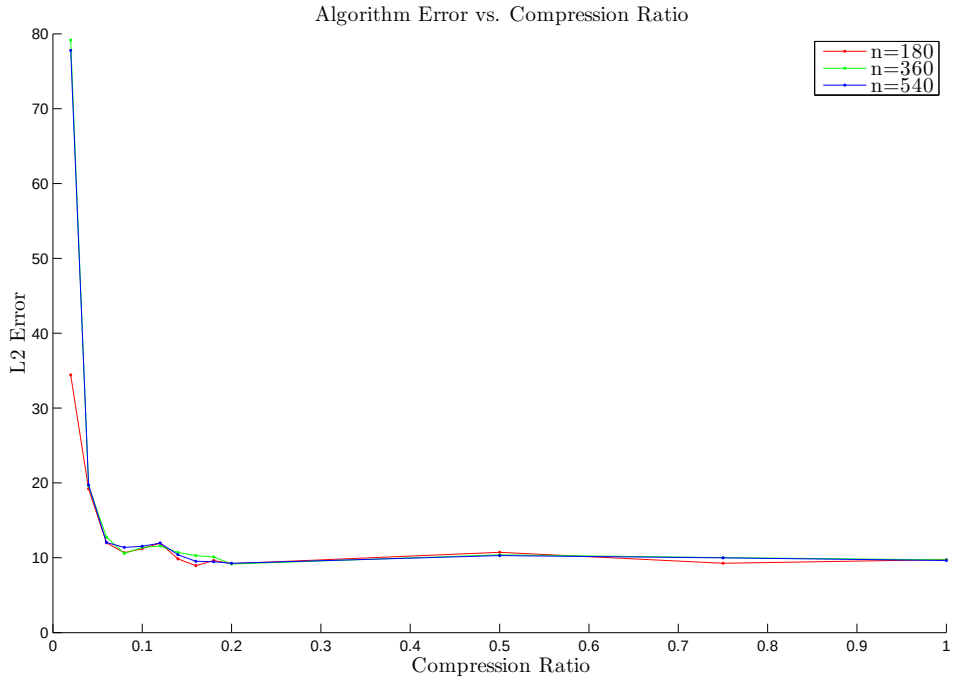


Figure 3-10: The L2 error as function of compression level for a number of different histogram sizes N.

Chapter 4

Conclusion

In this thesis we explored the preservation of certain image properties under a dimensionality reducing process we term *folding*. The motivation for this work is size, weight, and power constrained settings where an efficient hardware implementation of folding is desired. Coupled with the feature recovery algorithms developed in this thesis, an imaging architecture with significant savings on resources, power, and computation can be achieved. The three image properties we considered are corners, translation, and rotation. We presented algorithms for acquiring these properties when only allowed access to lower-dimensional folded representations of the image. We subsequently then looked at the performance of these algorithms when operating on real images.

We found translation to notably be extremely well preserved under the folding process. Our algorithm was able to accurately register the translation between two images from their folded representations at low compression ratios, as low as 0.2% of the measurements of the original image. Similarly, rotation information was fairly well preserved, although not at quite the same levels of compression as with translation. Below a compression ratio of about 3-4%, performance degraded quickly. However, above this level, performance stabilized very quickly to give good results at these relativley low compression ratios.

Corner detection from folded representations presented more of a difficulty. Much lower levels of compression were necessitated to recover corners. A number of com-

pounding effects appear to cause corners to be lost throughout the folding process, especially at very high levels of dimensionality reduction. Moreover, there is the additional problem of the false corners that our algorithm returns. The implication of these false corners depends on the application, but nevertheless must be dealt with. It should be noted that the poorer performance of our algorithm at high levels of compression does not necessarily mean that corner information is not preserved under folding. We are actively looking into optimizations to our algorithm or alternative ways of detecting corners that can achieve higher performance. Also, many vision-based navigation algorithms that would utilize corners do not require all of the corners from a series of image frames. Rather, they only desire a small consistent subset of the corners that they track from frame to frame. Initial corner repeatability experiments indicate that our algorithm could still be useful in such scenarios even though it fails to acquire all of the corners.

4.1 Future Work

There are many natural extensions to the work in this thesis. The next step for the work in Chapter 2 is to connect our corner detection algorithm to a vision based navigation system capable of tracking corners. We could then measure the difference between the navigation solution from our features and that of uncompressed corner detection. This would provide a quantifiable performance loss number to compare against the power savings from the folding hardware architecture. Additionally, there are many other features used in vision-based navigation systems, notably SIFT [18], FAST [21], and SURF [1] features, just to name a few. It would be interesting to explore the preservation of some of these more complex features under folding.

The next step for the work in Chapter 3 is to connect our algorithms to a real series of translated or rotated imagery and test performance. This would introduce complexity in the form of noise, lens aberration, and imperfect translation or rotation. It would be interesting to quantify the effect they have and analyze the robustness of our techniques in the face of these changes. Another direction would be exploring how

to simultaneously detect both rotation and translation. In the uncompressed setting this is relatively easy, as detailed in Section 3.1. Once the rotation is determined from the DFT magnitudes, it can be reversed, yielding two images that are purely translated, which we register using phase correlation. It is initially unclear how to perform the analog in the folded domain. Given only access to the folded representations of the images, it seems difficult to figure out how to reverse the rotation on pixels that have already been linearly combined. A technique for doing so would yield the results of this thesis even more powerful to generalized translation and rotation problems.

4.2 Hardware Implementation

The relevance of the majority of this thesis is predicated on the assumption that an efficient hardware implementations of folding can be realized. Otherwise, the power savings from working with dimensionally reduced signals cannot be gained and operating on folded images would just add an unnecessary level of complexity. Fortunately, this assumption appears reasonable. Focal plane technology is rapidly advancing and the type of structured binning necessary to implement folding in the focal plane array seems imminently achievable. Additionally, Gupta, in [12], surveyed many other optical implementations of folding and came to a similar conclusion that this assumption is indeed valid.

Appendix A

Complexity Comparison between Folded and Traditional Phase Correlation

Preliminaries

The complexity of the building blocks of the algorithms are presented first.

- **2-d FFT:** Assuming row-column decomposition and use of the FFT, the 2-dimensional DFT calculation for an image of size N by N has complexity $O(N(N\log_2 N) + N(N\log_2 N)) = O(N^2\log_2 N^2) = \boldsymbol{O(N^2\log N)}$ [17, p. 163].

- **Phase Correlation:** Correlation of two N by N images involves 2 forward DFTs, 1 inverse DFT, and element by element multiplication. It has complexity $O(2N^2\log N + N^2 + N^2\log N) = \boldsymbol{O(N^2\log N)}$

- **Correlation Peak Selection:** Finding the maximal peak in the output of circular correlation is $\boldsymbol{O(N^2)}$.

- **Folding:** The complexity of folding for an image of size N by N is $\boldsymbol{O(N^2)}$. One must simply add each image pixel to it's hashed bin value.

- **CRT decoding:** Chinese remainder theorem decoding of 2 residuals modulo p_1, p_2 involves finding 2 modular multiplicative inverses, ($p_1^{-1} \pmod{p2}$) and ($p_2^{-1} \pmod{p1}$). Assuming the Extended Euclidean algorithm is used to find the multiplicative inverses, the complexity of doing so is $O(\log(min(p_1, p_2)))$. The overall CRT decoding algorithm then involves multiplying these inverses by the residuals and summing them. Thus the overall complexity is $O(2\log(min(p_1, p_2)) + 2O(1) + 1O(1)) = \boldsymbol{O(\log(min(p_1, p_2)))}$. [6, p. 860]

Comparison

We can now derive the complexities of the corresponding algorithms. Again, assume 2 inputs images of size N by N.

- **Phase Correlation on full input images:** This involves correlating the two full input images and searching for the maximal peak. According to the background above, the complexity of this operation is $O(N^2 \log N + N^2) = \boldsymbol{O(N^2 \log N)}$.

- **Folded Algorithm:** The folded algorithm involves folding each input image twice, to size p_{1a} x p_{1b} and p_{2a} x p_{2b}. Then correlation and peak selection is carried out on the two sets of folded images. Finally, CRT decoding is performed in each dimension on the location of the identified peak in the two folds. We will assume $p_{1a} \approx p_{2a} \approx p_{1b} \approx p_{2b} = p$. This is good practice such that image content in both folds is jointly maximized. Moreover, we will assume we are using the minimum p necessary to decode the peak at any point in the image. For this, the necessary requirement is that the product of the fold sizes is greater than N, $p^2 > N$. This implies that the minimum necessary $p \propto \sqrt{N}$. The complexity of

this step is:

$$
\begin{aligned}
&= O(2N^2 + 2p^2 \log p + 2p^2 + 4 \log p) \\
&= O(2N^2 + p^2 \log p) \\
&= O(2N^2 + (\sqrt{N})^2 \log \sqrt{N}) \\
&= O(2N^2 + N \log \sqrt{N}) \\
&= \boldsymbol{O(N^2)}
\end{aligned}
$$

The folded algorithm brings the complexity down to $O(N^2)$ from $O(N^2 \log N)$.

Note that this assumes the smallest possible p is able to decode the translation amount (i.e. identify the correct peaks in the folded correlation matrices). As Figure 3-5 shows, we have experimentally verified this to be the case for even relatively large shift amounts with $N = 1024$, however the situation could change for different N. The compression ratio at the minimum possible p is $\approx \frac{p^2}{N^2} = \frac{(\sqrt{N})^2}{N^2} = \frac{1}{N}$. This gets very small for large N and it could be the case that at some large enough N the minimum p does not work and the value of p must increase ($p > \sqrt{N}$) for the folded algorithm to work, thus breaking the complexity arguments presented above. We have to experimentally verify in order to know.

Bibliography

[1] H. Bay, A. Ess, T. Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110:346–359, 2008.

[2] S. Becker, J. Bobin, and E.J. Candes. Nesta: A fast and accurate first-order method for sparse recovery. `http://www-stat.stanford.edu/~candes/nesta/NESTA_v1.1.zip`.

[3] S. Becker, J. Bobin, and E.J. Candes. Nesta: A fast and accurate first-order method for sparse recovery. Technical report, California Institute of Technology, April 2009.

[4] François Bourgeois and Jean-Claude Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Commun. ACM*, 14(12):802–804, December 1971.

[5] E.J. Candès, J.K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, August 2006.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[7] E. De Castro and C. Morandi. Registration of translated and rotated images using finite fourier transforms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(5):700 –703, sept. 1987.

[8] D.L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289 –1306, april 2006.

[9] M.F. Duarte, M.A. Davenport, D. Takhar, J.N. Laska, Ting Sun, K.F. Kelly, and R.G. Baraniuk. Single-pixel imaging via compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):83 –91, march 2008.

[10] A. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937 –947, june 2010.

[11] Rishi Gupta, Piotr Indyk, Eric Price, and Yaron Rachlin. Compressive sensing with local geometric features. In *Proceedings of the 27th annual ACM symposium*

on Computational geometry, SoCG '11, pages 87–96, New York, NY, USA, 2011. ACM.

[12] Rishi Vijay Gupta. A compressive sensing algorithm for attitude determination. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2011.

[13] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 31AUG- 02SEP 1988.

[14] C. D. Kuglin and D. C. Hines. The phase correlation image alignment method. *IEEE Conference on Cybernetics and Society*, pages 163–165, 1975.

[15] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[16] J.P. Lewis. Fast normalized cross-correlation. `http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html`.

[17] Jae S. Lim. *Two-dimensional signal and image processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[18] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, September 1999. IEEE Computer Society.

[19] M. McGuire. An image registration technique for recovering rotation, scale and translation parameters. `http://cs.brown.edu/people/morgan/papers/(McGuire%2098)%20Registration.pdf`, 1998.

[20] J. Romberg. Imaging via compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):14 –20, march 2008.

[21] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[22] J.A. Tropp and S.J. Wright. Computational methods for sparse solution of linear inverse problems. *Proceedings of the IEEE*, 98(6):948 –958, june 2010.