School of Electronics and Computer Science
Faculty of Engineering, Sciences and Mathematics
University of Southampton

Simon Goodman

May 12th 2005

X-ray Image Mosaics

Project supervisor: Dr. K. Martinez
Second examiner: Dr. C. J. Saunders

A project report submitted for the award of
BSc Computer Science

Abstract

An investigation into mosaicking algorithms for use with x-ray images of paintings. Many art galleries wish to have available x-rays of their paintings. Due to size limitations in the x-ray process, most paintings must be x-rayed in small sections. This produces a set of images that must then be joined, a process this project had aimed to automate. This report presents a mosaicking framework, which may be useful in other mosaicking problems. Two join methods are also presented and analysed. The first of these methods is a brute force approach that attempts to join the images in all possible ways, selecting the best for the final join. The second approach attempts to identify points in the images that contain ‘interesting’ features. It then uses these features to match against the second image. Once this matching is done with a number of feature points, the data produced is analysed in multiple stages until a final join-point selection is made. Once the join-points are selected, the mosaicking framework uses the selected join-points to merge the images together.

Contents

Acknowledgements

I would like to thank Dr. Kirk Martinez for his role as project supervisor and Dr. Craig Saunders for his role as second examiner.

1 Introduction

Many art galleries make use of x-ray imaging to study the materials, internal structure, and condition of a painting [1]. The x-ray images are acquired by placing x-ray film on top of a painting. As many paintings are larger than the greatest size of x-ray film available, it is often necessary to x-ray a painting using multiple sheets of film. These sheets of film are placed on top of the painting in rows and columns. Each of these rows and columns overlap the next. This allows the images to be joined by matching features found in the overlapping area, using them as a reference points for joining.

The current method for joining the x-ray images involves scanning the x-ray images into a computer, then manually joining them to form a mosaic of the entire painting. This manual joining is often done using nip2 [9], an image-processing program. The task of joining two images using this software involves the user identifying and selecting join-points within the images. As it is possible that there are small differences in the scale and orientation of a pair of images, it is necessary that two join-points be selected. The two join-points should be chosen from each end of the common edge. This reduces the error introduced to the image when correcting the scale and orientation differences between the images.

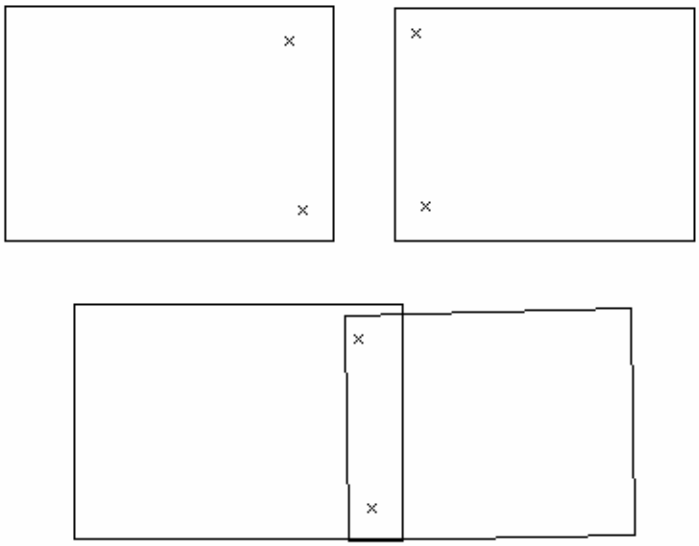

Figure 1. Illustration of join-point selection resulting in orientation and scale correction.

To mosaic the entire set of x-ray images the process of joining a pair of images is repeated multiple times. This produces increasingly large mosaics of sections of the painting, resulting ultimately in a complete mosaic.

This process of manually loading images, selecting join-points and merging images is very time consuming. Additionally, much care must be taken when selecting join-points in order to ensure the correctness of the final image.

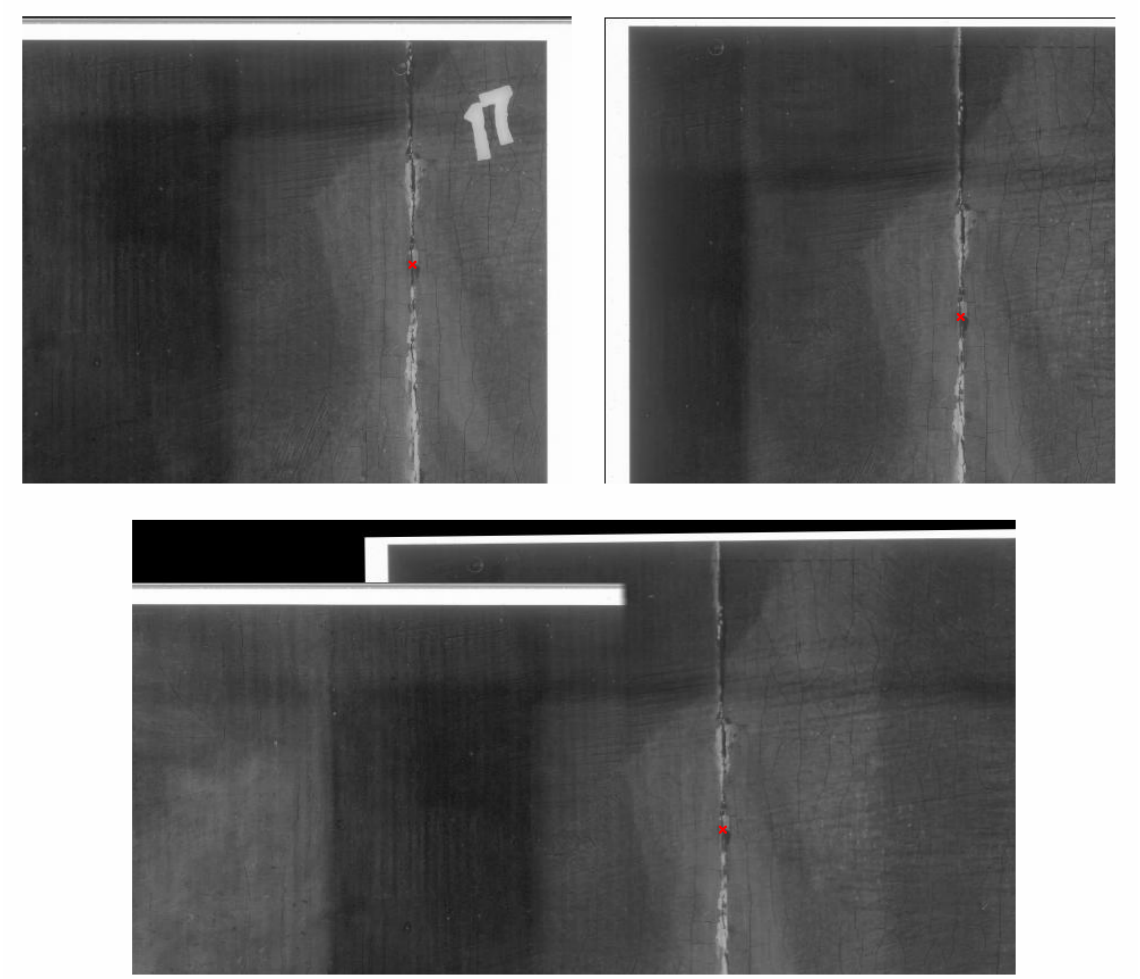

Figure 2. Image showing manual join-point selection (top) and the resulting merged image (bottom).

This project aims to automate all aspects of the mosaicking process. The resulting system should be capable of the following functions.

1. Joining a pair of images:
   Given a pair of join-points on each image, the system should be capable of merging a pair of images with a vertical (left-right join) or horizontal (top-bottom) common edge. The join-points from each image should be located in the same position within the resulting image.

2. Mosaicking a complete set of images:
   The system should be capable of taking a set of images and performing multiple joins building up a final image. The final image will contain all parts of the image correctly mosaicked.

3. Correctly identifying join-points:
   The system should be capable of locating points across a pair of images that correspond to a single feature within the original painting. These are then used to join the images.

Although the system aims to automate the process completely, it will still require a small amount of user interaction. This will consist of naming the images to be mosaicked in a

consistent manner. The images should contain at some point in their filename the numbers of the row and column to which the image belong. Additionally the row number should precede the column number, and there should be an identifiable separator between the two. This enables the system to mosaic the images in their correct positions.

The major focus of the project will be to create an algorithm that will correctly identify join-points across images. This is the most complicated section and will need researching. The other parts of the project should be relatively trivial to implement and are unlikely to require investigation.

One thing that will complicate the join-point identification algorithm is the three dimensional nature of the x-rays. This can cause features, such as nails, that are not part of the painting, but part of the support structure, to appear in different locations on different images. If one of these features is selected as a join-point, the result will be an incorrectly joined image. A pair of images illustrating this problem is shown below.

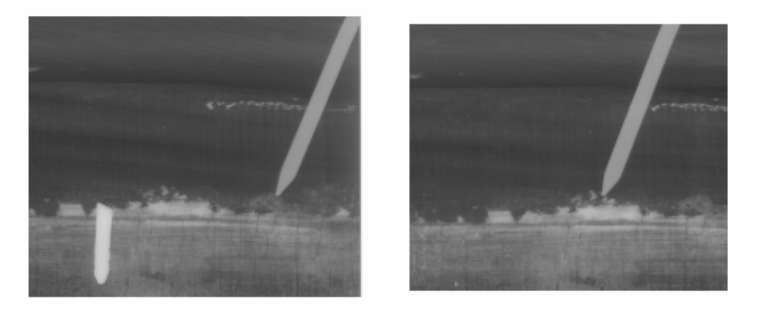

Figure 3. A nail appearing in different locations in different images.

1.1 Background Reading

Little work has been done on the specific problem of mosaicking x-ray images. Because of this, some of the information about mosaicking images in the literature may not be applicable to this specific task. This section reviews approaches to mosaicking in general. These may be directly relevant, or be the basis, or inspiration for an algorithm to mosaic x-ray images. Within the literature, the process of joining images together is known as registering. This terminology will be used throughout this section.

The most obvious way to automate the mosaicking process would be to replicate the manual process. Another approach is suggested by Niblack [2]. Instead of repeatedly joining image pairs, each image is taken in turn and registered to a common reference. The major drawback to this approach is the use of a common reference. It effectively requires a complete image of the painting. Each image is then placed over the complete image in order to determine its location within the mosaic. The complete image of the x-ray is what we are trying to produce and is therefore unavailable. One way this problem may be addressed would be to use a greyscale photographic image of the painting as the common reference. It is unclear whether the x-ray and photographic images would be similar enough to be used in this way. It is something that would require investigation before trying to use this approach. In addition, photographic images of the paintings may not be available even if the approach is feasible.

A number of join-point identification methods rely on three stages. The first stage is to calculate a rough positioning for the images to be joined. The second stage is to identify candidate join-points, i.e. features identified within the image. Finally, a way is needed to decide which candidate points correspond to one another in the two images to be joined.

Some key points identified by Zitová [3], in a survey of image registration methods, to take note of are as follows. When selecting possible features to be used as join-points it is preferable to choose “distinctive objects, which are frequently spread over the images and which are easily detectable”. In addition, the performance of the feature-matching algorithm should not be affected by features that do not have counterparts in the other image.

Zitová [3] recommends that feature based methods be used. However, if the images do not contain enough distinctive and easily detectable objects another approach should be used. It is suggested that area based methods are employed for medical images because of this. The images that are being worked with are not medical images, but because they are x-rays, the quality of the images may be comparable to medical images. Another method that is identified is that of cross correlation. Cross correlation can only exactly align mutually translated images. However, it can be successfully applied when slight rotation and scaling is present. As only a small rotation and scaling is present in the x-ray images, this method is a possible approach that can be explored.

Hsieh [4] proposed an approach that uses a wavelet transform to identify a number of feature points that can be used as a basis for image registration. Each feature point is an edge point whose “edge response is the maximum within a neighbourhood”. In order to estimate the orientation difference between the images, an “angle histogram” is calculated. The approach can tolerate approximately 10% scaling variation and does not have to restrict the position and orientation of images. This approach was designed for use in images that may have a large orientation difference; the use of angle histograms in the method was used to solve this. The use of angle histograms would not be necessary for mosaicking x-rays, as there is only a small difference in orientation between images. Locating feature points using local maxima is an approach that may be effective when working with x-rays of paintings.

A method that makes feature correspondences using optical flow estimation within small rectangular regions has been proposed by Chiba [5]. These regions are selected by identifying prominent features in the first image from the image derivative. Point correspondence is obtained by interpolating the results of four patches (13x13 regions) that are near to each other, bilinearly, in sub-pixel order. False correspondences are discarded by measuring the cross correlation coefficients. This approach again uses edge data (the image derivative) for locating feature points within an image.

A fast stitching algorithm using two complementary methods is proposed by Hsieh [6]. The first method used is edge alignment, which provides initial image translation data. This involves checking the consistency of edge positions between images and is capable of overcoming large displacement and lighting variations between images. The complementary method is a correspondence based approach that estimates desired parameters from a set of correspondences using a feature extraction scheme and a correspondence building method.

Control point and intensity are two basic features used separately for image registration. Li and Leung [7] proposed an exact maximum likelihood (EML) method that combines these two methods. Their method maximises the likelihood function based on control point and intensity to estimate the registration parameters. If data from either the control point analysis or intensity analysis is unavailable, or can be determined to be of poor quality, this method will still successfully register the images using the intensity or control point data respectively.

After this initial review of the background literature, the best approach appears to involve using edge data to locate points of interest, then correlating these points of interest. Hsieh's edge alignment method (from his fast stitching algorithm) may be useful for providing an initial estimate of the relative displacement between two images that are to be joined. As my research did not reveal any algorithms that were directly related to the problem of mosaicking x-rays of paintings, its use is limited to providing guidance as to how image registration can be approached in a more general manner.

2 Design Strategies

This section describes all the details related to the design, implementation and testing of the mosaicking system. After explaining the programming environment selection, it describes the architectural design, a mosaicking framework, and finally two methods of join-point selection.

2.1 Programming Environment Selection

The selection of the programming environment was based upon the decision to take advantage of the VIPS API [8]. VIPS provides a wide variety of image processing functions ready for use. Versions are available for both the Windows™ and Linux platforms, however only binaries are provided for Windows™ whereas the source code, header files and libraries are provided for Linux. Because of this Linux was chosen as the preferred platform. APIs are provided for both the C and C++ programming languages. It was decided that using C was the best choice for two reasons, firstly the object-oriented approach of C++ seemed unnecessary for the project, and secondly the C API for VIPS is better documented. The Linux distribution used for development was Fedora Core 2 running on a Pentium 4 1.5GHz machine with 512MB of RAM.

2.2 Architectural Design

At the start of the project, it was noted that the majority of the work would be creating and implementing a method for locating join-points on a single pair of images. Once a suitable method has been created for this, it is a simple task to apply the method repeatedly to a set of images in order to mosaic the entire x-ray. Because of this, it was decided that the project should be split into two sections; the first is a method for identifying join-points across a pair of images. The second is all other aspects of the project known as the mosaicking framework. It would be possible for the mosaicking framework to be used in all other mosaicking problems so long as the only distortions in the images are rotation and scaling. This part of the project was implemented first due to its relative simplicity, and because it allowed the join-point selection algorithms to be tested when they were implemented.

2.3 Implementation of the Mosaicking Framework

The mosaicking framework requires a two dimensional array of images sent as a parameter. It is assumed that the images are arranged in this array in the correct order for joining. The framework then decides in which order the images should be joined. There were two choices for implementation of this; perform all the left-right joins first then join the resulting rows using top-bottom joins (or vice-versa) or alternate between left-right and top-bottom joins building up larger and larger rectangular mosaics at each join.

The second approach was chosen as joining across a larger edge results in a smaller error after rotation and scaling, using the first approach the majority of the images are joined with only the height (or width) of a single image.

The framework is implemented by recursively calling a function that splits the array it is given in two, alternating between a left-right and top-bottom split on each recursion. The base case of the recursive algorithm is when the array it is passed consists of a single image, which is then returned. In all other cases, the algorithm divides the array into two then uses the mosaic function to mosaic each half. The algorithm then performs a join on the two images that were created by mosaicking each half of the array. The order of splitting and joining can be better understood by consulting Figure 4, The thickest line shows where the array is first split in two and joined, the lines get thinner and lighter as the recursion gets deeper.

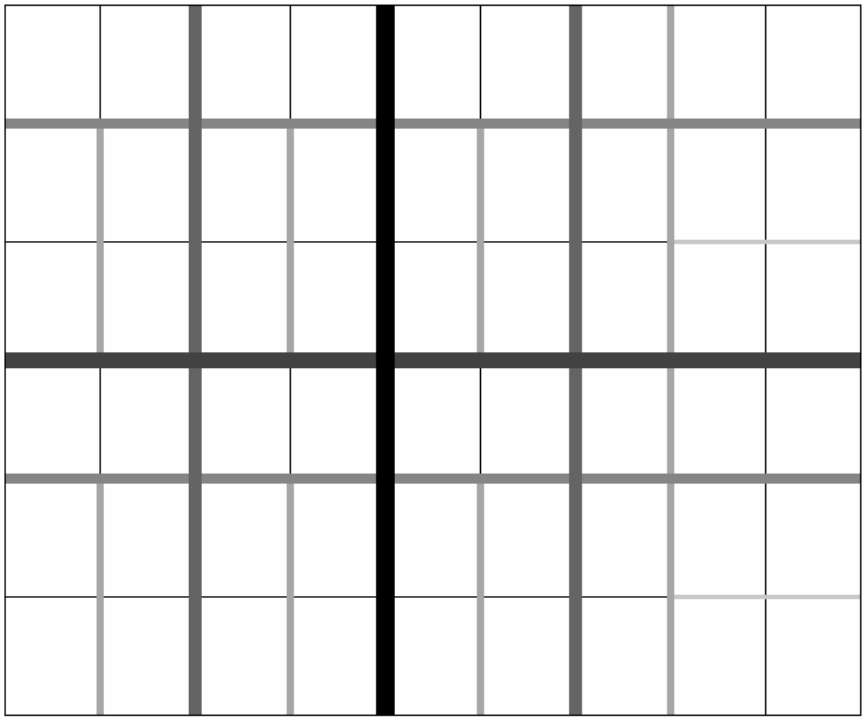

Figure 4. Illustration of the order in which a set of images are joined.

When the mosaicking framework was first implemented, the positions of the splits were different. Instead of splitting the mosaic at the midway points, it was split at the greatest power of two less than the width (or height) of the array of images to be mosaicked. This would mean that the majority of the image would be joined as squares, which is best. However, when it came to the right and bottom edge of the mosaic many images had to be joined into a single row or column before being attached to the rest of the mosaic. This made it harder to achieve a good join at those edges. Because of this, it was decided that it would be better simply to divide the arrays into two halves.

When joining an image, the mosaicking framework is passed two images and a pair of points on each image. In order to join these images using both the join-points, it may be necessary for the images to be rotated and scaled. The framework does this using the affine transformation function provided by VIPS [8]. The second image is scaled to match the first, and both images are rotated. This keeps the overall image straighter than rotating only one of the images, which is beneficial when using the images in a join later on in the mosaicking process. Black is used to fill the area around the image after it has been joined. The framework removes any rows or columns of pixels that are completely black at each edge of the joined image in order to prevent it building up.

2.3.1 Testing the Mosaicking Framework

Testing of the mosaicking framework was based upon using a stub for the join-point selection algorithm. The test stub was designed simply to generate a random point in each corner of the images to be joined.

To check the correctness of a single join within the framework, the randomly generated join-points were displayed on the console. Values for scaling and rotating the images were manually calculated from these join-points, and compared to values calculated by the framework. To ensure that the images were joined correctly according to the generated join-points, nip2 [9] was used to join the images manually and the results compared to the results from the framework.

Once I was satisfied that algorithms for left-right and top-bottom joins were correct, the rest of the framework was tested by mosaicking the entire test image using the random join-point stub. This was done to make sure that the framework was loading images in the correct order and placing images in the correct position relative to one another. This was verified by viewing the resulting image, and checking that each image was in the correct position within the mosaic. The mosaic produced is shown below.

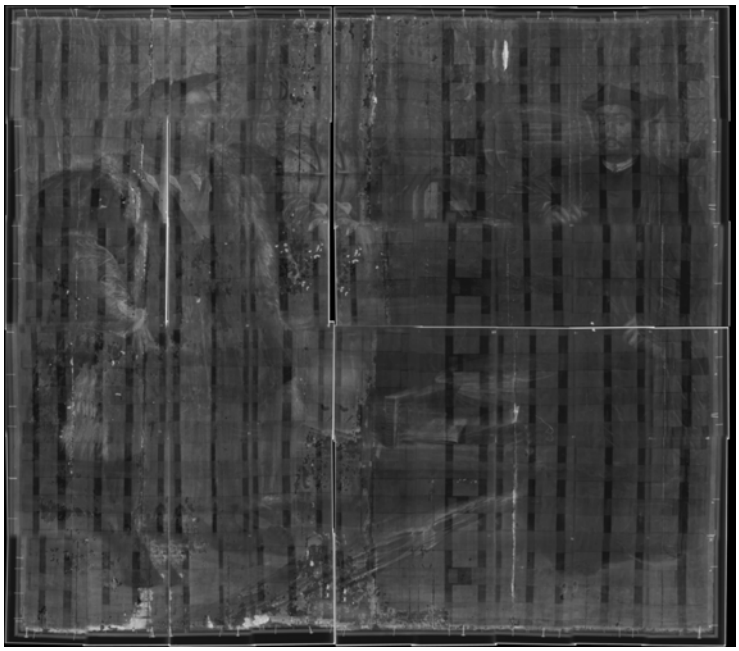

Figure 5. Mosaic created using random join-points.

Finally, to make sure that the mosaic had been divided and joined in the correct order the framework was modified to write all temporary images to disk. Each of the temporary images corresponds to a subsection of the entire mosaic. This allowed me to check the order in which each individual join was being performed by checking the order in which each temporary image was created on disk.

2.4 Join Method 1: Sliding Subtraction – Minimum Difference

The first attempt at solving the join-point selection problem was a simplistic approach. It consists of making a brute force join at the 'best' position. The idea of the algorithm is to position the second image over the first image in every possible position that is a likely candidate for joining the image. At each position, the algorithm calculates the average difference per pixel for all pixels within the overlapping region. The average difference per pixel is used because it is independent of the size of the overlapping area. If the total difference between all pixels were used, positioning the images with a smaller overlap would result in a lower total difference, purely because there are fewer pixels taken into account. The algorithm defines a likely candidate for joining the images as anything from a fifty to three hundred pixel overlap in the x direction for a left-right join and any position that results in less than one hundred pixels of the right hand image in the y direction hanging off the edge of the first image. The same values are used for a top-bottom join but with the x and y directions swapped.

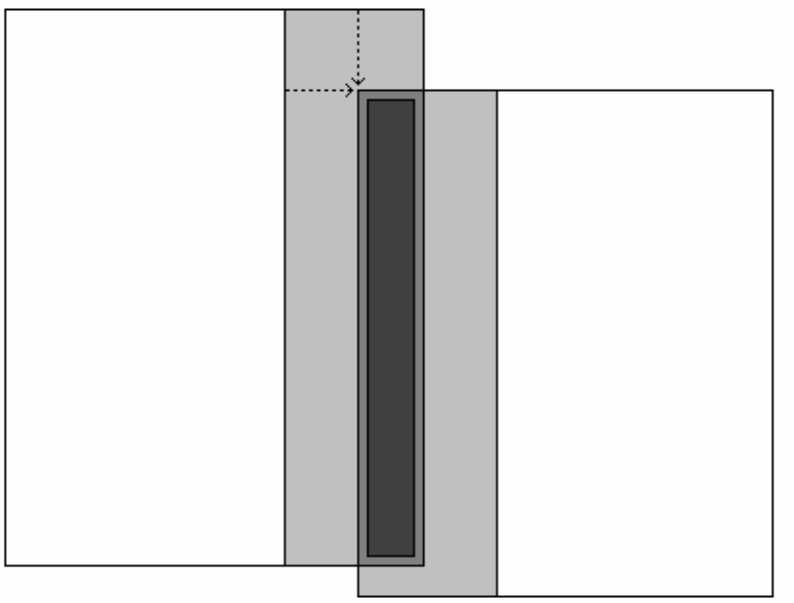

Figure 6. Image illustrating the area used for calculating pixel differences from the overlap region.

One point to consider is that as each join in the mosaicking process is performed, it may be necessary to add regions of black at the edge of the image. To remove any problems caused by this instead of differencing every pixel in the overlapping region a border is defined of twenty pixels at the edge of the overlapping region. The pixels that lie within this border are ignored, only those within the centre of the overlapping region are counted.

An important problem that this algorithm does not attempt to address is that of the relative scale, rotation and brightness differences between the images to be joined. Due to the nature of the algorithm, it will only perform correctly if the images have the same rotation, scale and brightness. However if the two images do have this property then the algorithm should correctly join all image pairs one hundred percent of the time, unfortunately this is not the case with the x-ray images the system is designed for.

Continuing in the brute force approach the algorithm could be extended to rotate, scale and adjust the brightness of one of the images to be joined in every possible combination until an exact (or almost exact) match is found. The amount of time taken to run an implementation of this proposed algorithm would be extremely large for even a single pair of images.

2.4.1 Testing of Join Method 1

To check the correctness of the implementation output was added describing what the algorithm was doing at run time. Variables were checked against manually calculated values for the bounds of the overlapping region.

This algorithm was tested for effectiveness by applying left-right joins and top-bottom joins to each possible pair of images from the test data provided (x-rays of ng1314 [10]). This consisted of 48 pairs of images to be joined left to right and 45 pairs of images to be joined top to bottom. The results were checked manually using the nip2 GUI, which is designed for use with the VIPS image processing system. In all cases, the images were incorrectly joined. This is most likely due to the shortcomings of the algorithm discussed in the previous section.

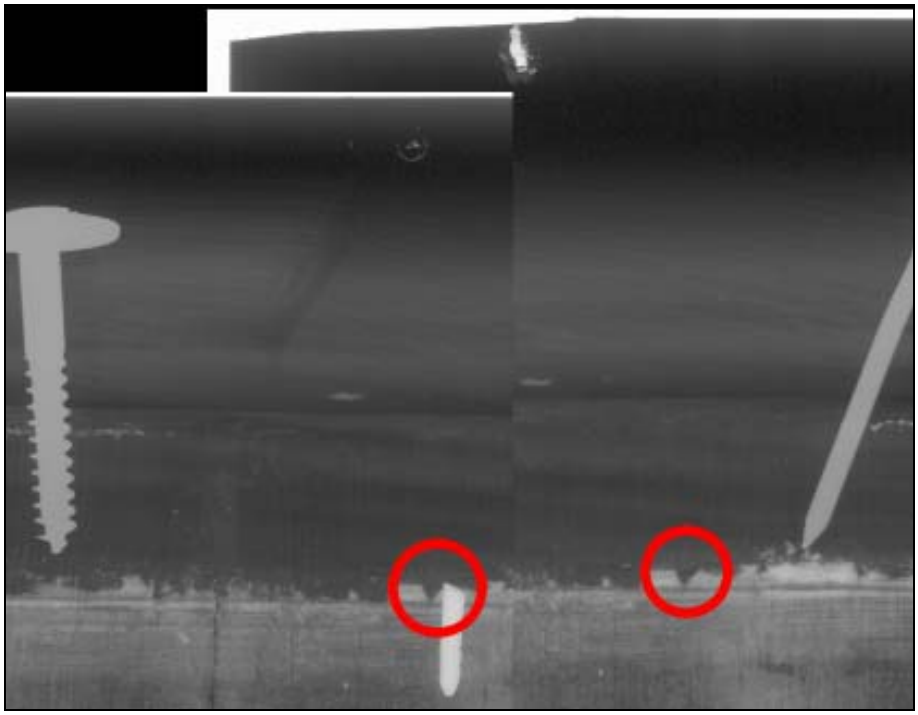

Figure 7. Example of a join made using the minimum difference method highlighting its failure.

In addition to failing to join the images correctly, each attempted join took a significant amount of time to run on the test system. The duration of each join ranged from 335 to 1056 seconds with a mean of 514 seconds for a left-right join, and a range of 216 to 494 seconds with a mean of 301 seconds for a top-bottom join.

Due to the failure of the algorithm to join even a single pair of images correctly, no attempt was made to mosaic the test image. It was decided that another approach was required; this second attempt is discussed in the next section.

2.5 Join Method 2: Edge Detected Cross Correlation

The second method that was tried for join-point selection involves many stages. This section provides a brief overview of the algorithm. Each stage is explained in detail in the following subsections.

First, two areas from each image (known as segments) are extracted and a sobel edge-detection operator applied. The algorithm then looks for points that seem to contain 'interesting' features in each of these segments and extracts a small region of the segment from around that point. An interesting feature is defined as a small region with a high edge response. Next, an attempt is made to match each of these regions to points in the corresponding segment from the opposite image.

After the initial matches are made, some analysis of the data is performed in order to choose the best join-points. The first step is to select only pairs of points that have regions matched to points from both the first to the second image and the second to the first image. Pairs of points are then grouped according to their proximity to one another. Next, they are grouped according to the relative displacement they would create if used as a single join-point for the two segments.

The final stage is analysing pairs of join-points to ensure that they are reasonable. A pair of join-points is discarded if they would require a large amount of scaling or rotation to be performed on the images. Finally, the remaining points are ranked depending on how many points have contributed to matching that point and the best pair of matches selected for joining.

Before giving the details of the algorithm, some definitions are required to make understanding and explanation of the algorithm simpler as confusion could otherwise easily occur.

- Point
  A single pixel location in an image defined by an x and y coordinate pair.
- Segment
  The part of the original image extracted in the first stage, in which points of interest are searched for and matched in.
- Region
  A small image (19 by 19 pixels) extracted from a segment containing an 'interesting' feature.
- Location
  A small area around a single point in the image. If two points are within the same location, they are considered close enough for correlating in stage 4 and grouping in stage 5.

2.5.1 Stage 1: Segment Extraction

The first stage of the algorithm is to extract the segments of the images in which to look for match points. This depends on whether we have two images that are to be joined top to bottom, or two images that are to be joined left to right.

Segment extraction is an important issue and may affect the success of the rest of the algorithm. If we extract too small a segment it may not be possible for the algorithm to find any matching points. If too large a segment is extracted the algorithm will become less reliable as only part of the segment will be in the overlapping section of the image. In addition, increasing the segment size will increase the amount of time that it takes the algorithm to join the images.

Another factor that must be considered at this stage is that some of the images that are to be joined may have a border. This may be as part of the original image, or a black border that has been added from previous joins. A small gap is left between the edge of the segment and the edge of the image. This increases the likelihood that the segment only contains part of the image and not any border. Experimentation has shown that a region of 500 by 350 pixels is best, the 500-pixel edge being along the edge of the join. This can be seen in below in figures 8 and 9.

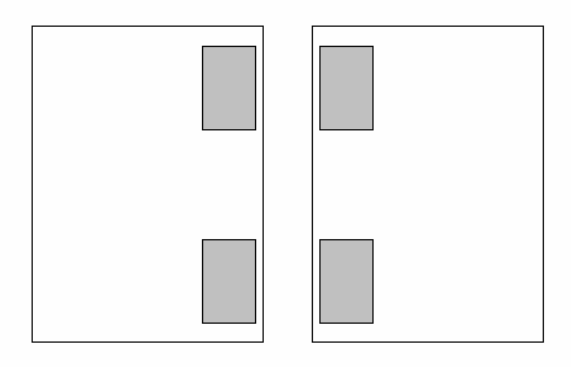

Figure 8. Image segment positions (left-right join).

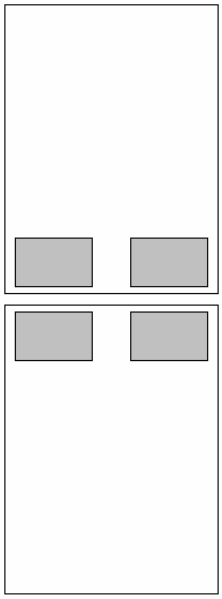

Figure 9. Image segment positions (top-bottom join).

It is likely that the images to be joined will need to be rotated and scaled slightly in order to be able to make the join. Because of this, the segments are extracted from the edges of the image. This creates a greater distance between the two tie points that will be ultimately selected, and thus allows for greater accuracy in these two transformations.

Once the four segments have been extracted from the images, a sobel edge-detection operator is applied to the images. The first reason for this is it allows us to identify regions with 'interesting' features to be extracted in the next stage. The second reason is it allows comparison of the two images independently of any global brightness difference between the two images; this is used in a later stage. All following calculations are done using the edge-detected images.

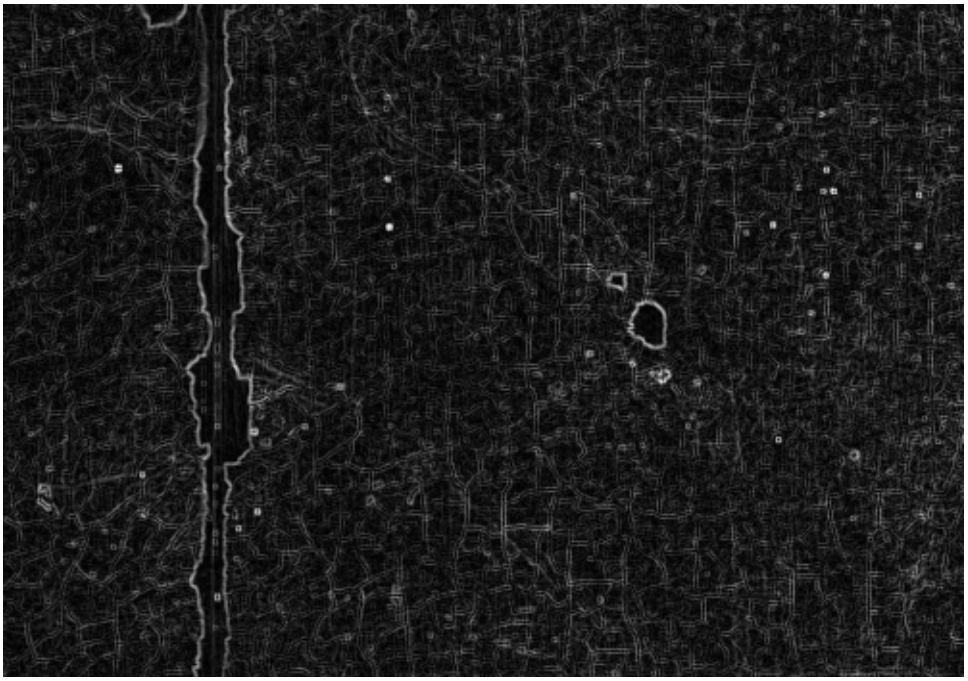

Figure 10. Example segment after applying a sobel edge-detection operator.

2.5.2 Stage 2: Extraction of Regional Maxima

Once the segments have been extracted from each of the images to be joined, regions in each segment that contain a large amount of edge information are located. The idea is to find regions of 19 by 19 pixels that have the greatest value when the edge data, for each pixel within the region, is summed. These regions correspond to areas of high contrast within the original image and are likely to correspond to locations in the image that have features useful for matching.

Initially the number of regions extracted is set to 16, however if in any later stage it is found that none of the extracted regions are useable, the algorithm returns to this point and increases the number of regions extracted. With each iteration, the number of regions extracted doubles. A final cut-off is set at 128 regions and if the image can still not be joined the algorithm returns failure. This is for two reasons. Firstly, initial testing found that images joined using fewer regions were more likely to be correct. This means extracting a large number of regions should be avoided unless it is necessary. Secondly, the computational complexity of the algorithm is related to the number of regions extracted and has a significant effect on the time it takes the algorithm to run.

For simplicity of implementation, a 19 by 19 convolution template is used instead of manually calculating the sum of the edge values for each possible region within a segment. All the values of the convolution template are set to one producing the same

effect as summing all the values of pixels within each possible region and storing the result at the centre point of that region. The locations of pixels within the convolved image having the highest values are used as the centre of the regions to be extracted. A set of images is then extracted from each edge segment for use in the next stage of the algorithm.

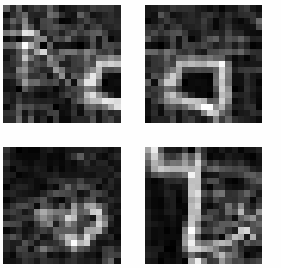

Figure 11. Examples of regions extracted for later matching.

In parts of a segment where there is a high edge response, it is possible that two or more neighbouring pixels will be selected for use as a region. This results in us having multiple regions all representing a slightly different part of the same image feature. In order to prevent the same location being selected many times a stepping values is used. This stepping value is defined as half the width of a region. Instead of checking every pixel location, we skip over n pixels at a time; where n is equal to the stepping value. The usefulness of the stepping value can be seen by looking at the top two regions in figure 11. These two images both contain part of the same feature; these were the only two from the complete set of regions that contained this feature. If no stepping value is used many more of the regions would have included this same feature at slightly different positions in each.

2.5.3 Stage 3: Matching Regions in Corresponding Segments

There is now a set of regions extracted from each of the four image segments. This next stage takes each of the regions for a segment, and then calculates the locations at which it best matches in the corresponding segment, i.e. the segment on the opposite side of the join. This is achieved by placing the region on top of the opposing segment in all possible positions. For each position, we calculate the total of the difference between the pixels in the region and the pixels in the segment. We store this difference along with the location of the match for the sixteen matches with the lowest differences. This is done for each region within each segment. Experimentation seemed to indicate that using sixteen matches was a good number to choose. This stage is the most computationally expensive; its performance is affected by the size and number of regions extracted in stage 2, and the size of the segments extracted in stage 1.

2.5.4 Stage 4: Correlation of Selected Maxima

We now have a set of points in each segment, with each point linked to a set of points in the corresponding segment. In order to select only the best matches the algorithm correlates the information from each pair of segments. To do this it looks through points in the first segment that have been linked to the second segment. If it finds that the

location in the second segment has also been linked to the location in the first segment the pair of points are kept. If there is no mutual match then the locations are discarded.

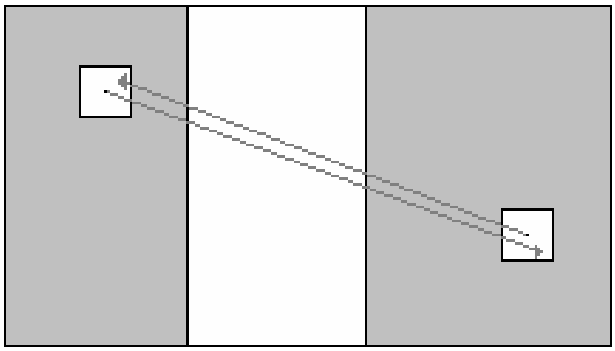

Figure 12. Illustration of correlation using locations around match points.

As it is unlikely that there has been an exact match between two points, both segments points are considered to match if they are in the same location. Two points are considered in the same location if they are only a few (less than eight) pixels away in both the x and y direction.

There is a choice of two point mappings, that from the first to the second segment and that from the second to the first segment. The mapping with the lowest difference calculated in stage 3 is the mapping that we keep and the other is discarded.

2.5.5 Stage 5: Grouping of Correlated Points by Location

The correlated points are further reduced in number by grouping together mappings between segments that are near each other. This is done in a similar fashion to stage four by defining a location as an area around each point. If there is more than one mapping where both points are in the same location they are combined into one. A record (point count) is kept of how many mappings have been combined into a single mapping. Again, we have a choice of which mapping within the region is the best one to use for the final join and as in stage four the mapping with the lowest difference from stage 3 is used to make this choice.

2.5.6 Stage 6: Grouping of Correlated Points by Relative Location

Mappings between segments can further be grouped according to the difference in the x and y coordinates between the location of the point in the first segment and the location of the point in the second segment. This is effectively grouping the pixels according to how they would displace the first segment against the first, if they were chosen as a join-point.

This grouping can be seen more clearly by looking at figures 13 and 14. Figure 13 shows a set of mappings that will be grouped by this part of the algorithm. Figure 14 shows that all these mappings result in the two segments being joined in the same way. As in previous stages this displacement needs not be exact, a small difference is acceptable.

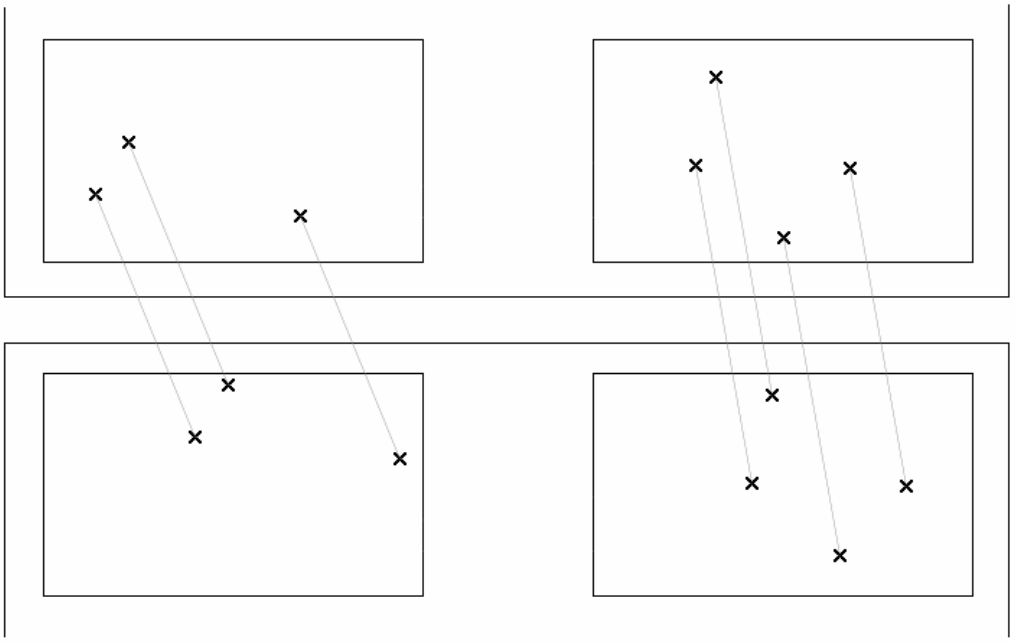

Figure 13. Illustration showing how many pairs of points result in the same displacement.

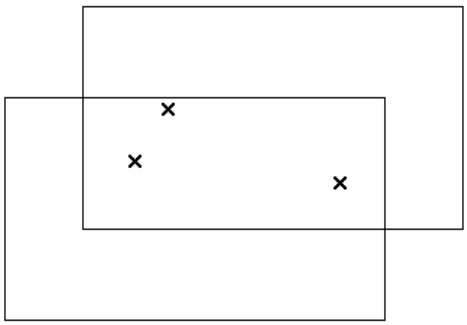

Figure 14. Result of joining the left segments using the join-points from figure 13.

The segments will be slightly scaled and rotated relative to one another when they are joined, but this is compensated for by allowing a little leeway in the exact locations of the points.

A record is kept of how many mappings have been joined together in this manner. Also recorded is the number of mappings grouped in this stage, this is known as the delta count. Finally, the number of points that have contributed to the mapping by summing all the point counts (from stage 5) is recorded. As in previous stages, the mapping with the lowest difference is kept to keep track of the best match from the mappings that have been grouped.

2.5.7 Stage 7: Removal of Obviously Incorrect Point Pairs

The next stage discards obviously incorrect data. As an attempt is made to make the x-rays line up as best as possible when they are taken, we know that there will be only a small difference in the rotation and scaling of the two images to be joined. This knowledge can be used to remove some sets of join-points.

Taking a top-bottom join as an example, we have a set of mappings for points on the left hand side of the image and a set of mappings for points on the right hand side of the image. In this stage, we take each possible pairing of mappings for the left and right hand sides of the image. We then calculate how much we must scale and rotate the right hand image in order to join them according to the current paring of points. If the angle of rotation is too big (3.5 degrees is used) then the pairing is discarded. Also if the scale factor (1.1 is used) is too large then the pairing is discarded. Finally, if the pairing results in the bottom image being moved to the left or to the right by a significant amount (150 pixels is used) then the pairing is discarded. All the pairings that are not discarded are kept and analysed in the next stage when the final join-points are selected.

This is the only stage of the algorithm along with the segment extraction in stage 1 that has different code depending on whether a left to right or top to bottom join is being performed. All other stages have code that can be used independently of the direction of the join.

Figure 15 shows how the orientation and scale difference is calculated from a set of join-points. The angle between the points is in each image is calculated using trigonometry. The length of the line between the two points is calculated using Pythagoras' theorem. The angle of rotation is calculated by subtracting the two angles; the scale difference is calculated by dividing the two distances between each pair of points.

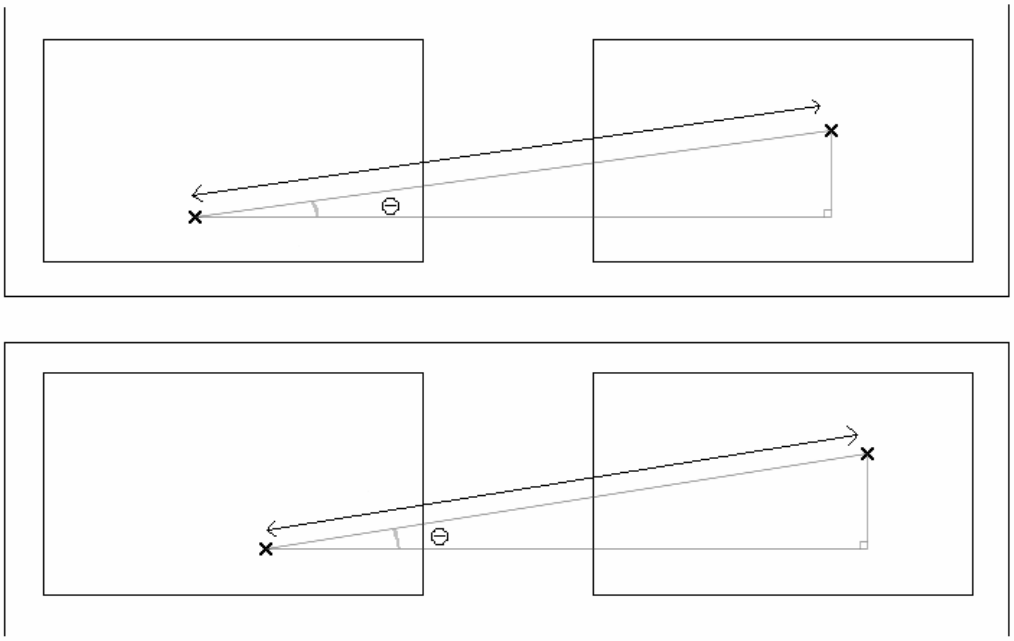

Figure 15. Image showing how scale and rotation differences between images are calculated from a set of join-points.

2.5.8 Stage 8: Selection of Final Join-points

At this stage, there is a list of reasonable match pairs along with information about how many points and locations have contributed to them and the total difference between the two regions about those points. This gives us enough information to be able to select the best pairing from those that have not been discarded in the previous stage. A simple

formula is applied to each pairing and the pairing with the lowest value is the one that is selected. This formula is:

$$\frac{\text{diff\_a} + \text{diff\_b}}{\text{point\_count} \times \text{delta\_count}}$$

- diff_a is the difference between the two regions about the mapping used for the left of the join.
- diff_b is the difference between the two regions about the mapping used for the right of the join.
- point_count is the total number of points used to make the pairing.
- delta_count is the total number of locations used to make the pairing.

This formula makes use of the information in the following ways. It prefers pairings of points with a small difference about their regions. It prefers pairings with a high point count and thus a large consensus that there is a match at that point. Finally, it prefers pairings with a high delta_count and a large consensus that displacing each segment by that amount is a good choice. Once this stage is completed, the selected join-points are returned to the mosaicking framework.

2.5.9 Testing of Join Method 2

There are two aspects to the testing of the second join method. The first is testing the correctness of the implementation. The algorithm was implemented incrementally one stage at a time. After each stage had been programmed, it was tested for correctness before beginning implementation of the next stage. The second aspect was to test the effectiveness of the algorithms join-point identification. Each of these aspects is discussed in turn in the following sections.

2.5.9.1 Testing the correctness of the implementation

Stage 1 of the algorithm (segment extraction) was tested by writing the extracted image segments to disk. These images were compared to the original images from which they were extracted to verify that the correct parts of the images were extracted. The sobel edge-detection operator was then tested by writing the edge-detected images to disk. These were compared to images created using nip2 from the extracted segment created in the first part of testing.

The edge-detected images are then averaged in stage 2 to locate regions with a large amount of edge data. These images were loaded into nip2 and the maximum points of the image were located using functions provided by nip2. The coordinates of the points identified as maxima by the algorithm were output and compared to the coordinates of the points found using nip2. Additionally, the extracted regions were written to disk and a check was done to ensure that they corresponded to the area in the image located at the coordinates output on the console.

Due to the amount of data generated in stage 3 when matching regions, exhaustive testing of all the results was not possible. However, a subset of the regions extracted in

stage 2 was chosen for verification. The coordinates of all the matched points were output, and the region was visually compared to those points in the original image. All the points looked similar to the selected region, and this was considered enough evidence that the algorithm was operating correctly.

All the following stages are based on a numerical analysis of data produced in the previous stages. Because of this, the remainder of the algorithm was checked for correctness by using textual output from the program. In each stage, the program described which points it was correlating and grouping along with any information related to the decisions it made. This consisted of the relative displacement of points for stage 6, and the scale and orientation differences in stage 7. Output for stage 8 included the calculated rank for each set of points. As each of these stages of the algorithm was implemented, the output produced was analysed by hand in order to verify that the stages were functioning correctly.

The final stage of testing was to make sure that the images were being joined using the correct points. This was done by outputting the coordinates of the selected join-points and performing a manual join of the images using those points. The manually produced image was then compared to the programmatically produced image. Although this test had been done during the testing stage of the mosaicking framework, it was felt that it should be done again after implementing a proper join-point selection algorithm. Additionally, when performing the manual join, the selected points were visually analysed to make sure that they corresponded to some sort of ‘interesting’ feature.

2.5.9.2 Testing the join method for effectiveness

Testing for effectiveness was undertaken in much the same way as for the first join-points selection algorithm. The algorithm was run on all possible pairs of images from the test image. This consisted of 48 left-right joins and 45 top-bottom joins. All the resulting images were viewed using nip2 and visually verified for correctness.

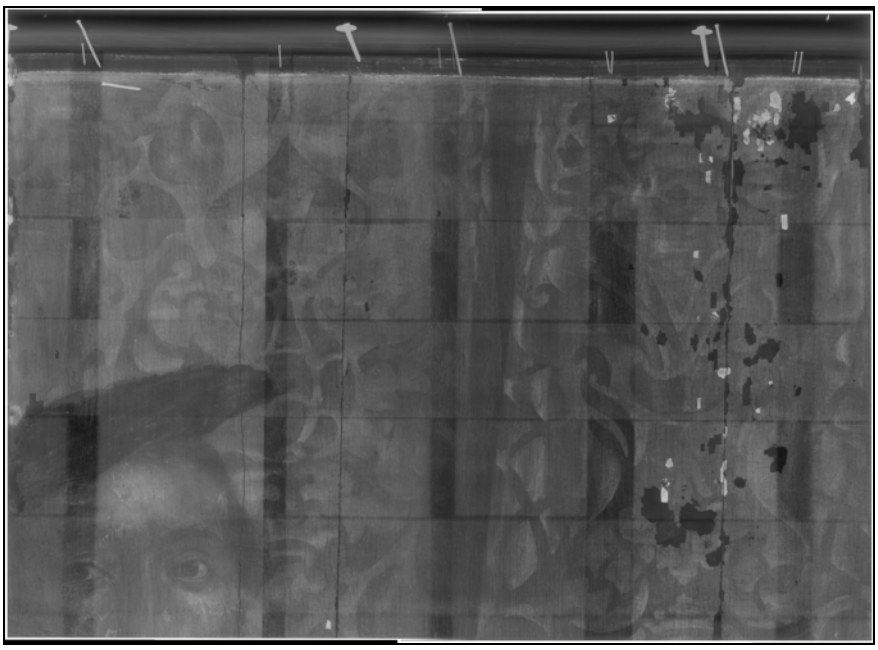

Figure 16. Example left-right join.

44 of the left-right joins were correct, a 92% success rate with 0 failed joins. 32 of the top-bottom joins were correct, a 71% success rate with 2 failed joins. A top-bottom join took on average 480 seconds to complete, ranging from 83 to 2321 seconds. Left-right joins took on average 167 seconds to complete, ranging from 82 to 1233 seconds. An example of a correct left-right join is shown below.

After finding that both left-right joins and top-bottom joins achieved a good success rate, an attempt was made to mosaic the entire test image. The program was unable to perform three of the joins; these had to be done manually when prompted. The resulting image, shown below, had many errors.

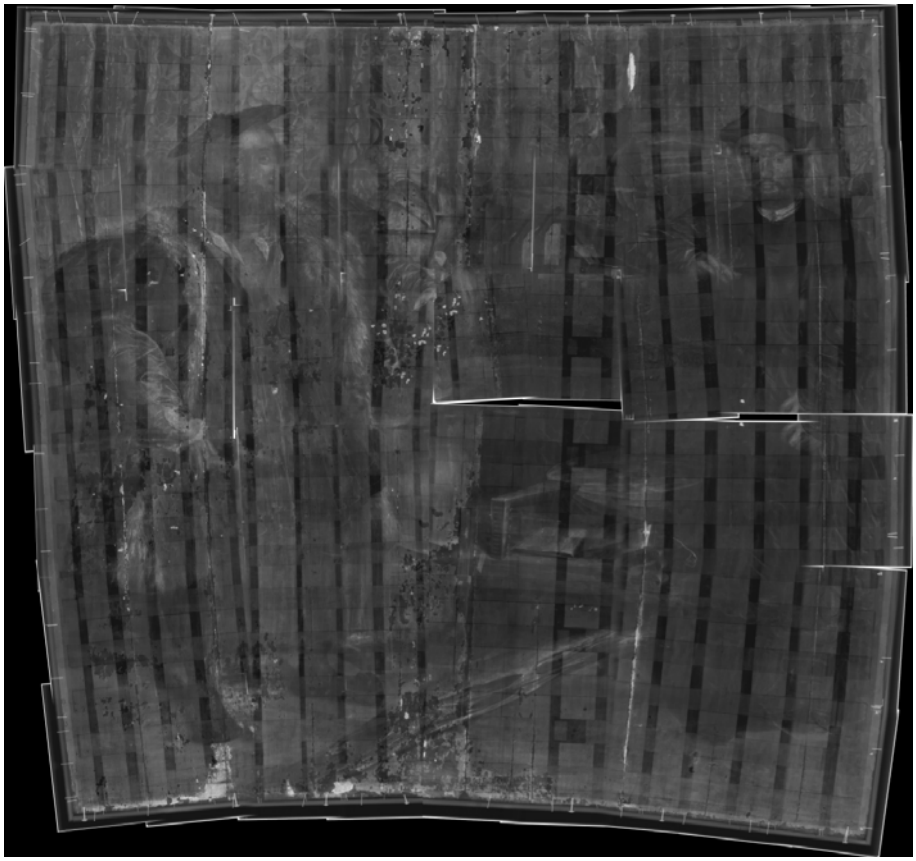

Figure 17. Result of mosaicking ng1314 [10].

Once a join has been made incorrectly in the mosaic, the program is unable to join the part of the mosaic with the error to any other part correctly. Because of this, error spreads across the image resulting in a failed mosaicking attempt.

Because the errors spread across the mosaic, it was decided that the order in which the images are joined should be changed to see if it resulted in an improvement. Each row of images was mosaicked first, and then the rows mosaicked to produce the final image. Rows 3, 5 and 6 of the image were mosaicked without error. Rows 1 and 2 both had two incorrect joins, and row 4 had a single error. When the rows were mosaicked to produce the complete image, none of the joins were correct. This was expected as each join involved at least one incorrectly joined row. The exception was row 5 and 6; however, this join was still made incorrectly. The resulting images are shown below.

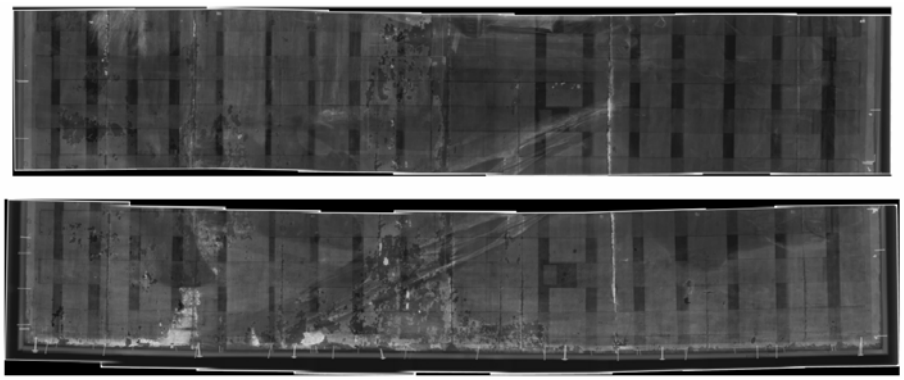

Figure 18. Rows five and six of ng1314 [10] correctly mosaicked.

Figure 19. Result of mosaicking ng1314 [10], joining complete rows first.

3 Conclusions

The three aspects of the mosaicking process that this project aimed to automate were:

1. Joining a pair of images
2. Mosaicking a complete set of images
3. Correctly identifying join-points

Joining a pair of images has been achieved; this part of the project was the easiest. All that was required was to calculate the appropriate affine transformations to apply from a set of join-points. These transformations were then applied to the images and the images merged using functions provided by VIPS [8].

Mosaicking a complete set of images is the part of the project that has not been completed. This is largely due to locating appropriate areas of images in which to search for feature points. As more and more images are mosaicked, the area of an image that contains information moves further away from the edge. The join-point selection algorithm extracts an area that is a fixed distance from the edge of the image. If the algorithm looked for an area that contained real image information, the algorithm would probably have been far more successful.

Correctly identifying join-points has been successful. Testing of the join-point identification algorithm has shown that join-points can be identified for two individual images 82 percent of the time. The algorithm can be improved to make it even more likely to succeed, and to determine when the join-points it has identified are incorrect. These improvements are discussed in the following section. The algorithm only fails when applied to images that are the product of mosaicking a subsection of the entire mosaic. This is for the reasons discussed in the previous paragraph.

Although 82 percent of the test images were joined correctly, the join-point selection algorithm needs to be improved still. Currently, the algorithm is considered to have joined a pair of images correctly even if there is a small error (join-points are a few pixels out). This is something that would need improving before the algorithm would be of use.

Overall a viable approach to the problem has been developed. This approach can be developed further and could be successfully used to mosaic x-ray images of paintings. These improvements are the topic of the next section.

4 Future Improvements

A number of modifications, additions and optimizations can be made to both the mosaicking framework and the join-point selection algorithm. These are discussed in the following sections. Additions and modifications are discussed in the next section and optimizations in the section following that.

4.1 Algorithm Improvements

Possible improvements to the mosaicking system are discussed in the following sections. All improvements to the join-point selection algorithm refer to the second approach to the problem.

4.1.1 Mosaicking Framework Improvements

At present, if a single join fails, the user is required to perform a join manually. It may be possible to work around a failure by changing the order in which the images are joined. Take as an example four images to be joined into a two by two mosaic. The mosaicking framework may attempt to join the top two and the bottom two images using a left-right join first, and then join the two resulting images using a top-bottom join. If one of these joins fails, the mosaicking framework could then attempt to join the left two images, and the right two images using a top-bottom join. The mosaic could then be completed by performing a left-right join on the two resulting images. The result would be the desired two by two mosaic. By changing the join order, the failed join can be avoided. This would increase the robustness of the algorithm; allowing a mosaic to be completed, even when using a join algorithm that does not succeed one hundred percent of the time.

Within the current mosaicking framework, each image is involved in multiple joins. Each join results in the original images being scaled and rotated. With each of these affine transformations, the images lose more of their original sharpness. This loss of detail is undesirable as the images are to be used for scientific analysis. It would be preferable if each image was only scaled and rotated once before being merged into the final mosaic. This could be implemented by keeping track of the affine transformations performed on each image throughout the mosaicking process. Once the mosaic is completed, a final output image could then be generated by transforming each image once and merging the results together.

A completely different approach to the way in which the images are mosaicked could be attempted. One way of doing this would be to take all neighbouring pairs of images within the mosaic and join them to each other. This would result in each image being mosaicked to four images (except at the edge of the mosaic). This would allow comparison of the joins to identify inconsistencies within the mosaic. If an image is mosaicked to one position in three of the joins and mosaicked differently on the fourth, this could be identified and the incorrect join ignored.

4.1.2 Join-point Selection Improvements

The final stage of the join-point selection algorithm involves ranking each set of join-points according to a formula. This formula involves terms for the difference in pixel values across the matched region, the number of points, and the number of locations contributing to the set of join-points. Currently the algorithm is not weighted in any way to make each factor contribute differently to the final selection. The formula could be modified to allow weighted contribution of each factor; and tested to see if it results in a performance improvement. More specifically, the number of locations contributing to a set of join-points could be given a greater weighting than the number of points.

One aspect of the join algorithm that needs further investigation is fine-tuning of the join-points. At present, although the join algorithm performs well, there is often an error of a pixel or two in the join-point selection. One way to approach this problem would be to look for local maxima within the location surrounding the join-points. Local maxima from both the original image data, and the image produced by applying an edge detection algorithm could be used. The points of these local maxima could then be used as the final join-point. This may or may not improve the accuracy of the join, but is something that needs investigating.

Although the join-point algorithm sometimes fails to join a pair of images in its current state, the success of the algorithm does not imply correctness of the join. A final check stage could be added to the algorithm that attempts to determine whether the images have been joined correctly or not. This could be implemented by finding the area of overlap in the joined image, and calculating how different the two images are. This would work best work on edge data to avoid the overall brightness of the images affecting the results. The calculation of how different the images are could be implemented in a similar method to the approach used in the first join-point algorithm, by finding the mean difference between pixel values in the two images. Some threshold value would need to be defined, above which the join is determined to be incorrect. If a join is found to be incorrect, the algorithm could return to stage eight and select a different set of join-points. If all sets of join-points result in an incorrect join then the algorithm would return failure.

One problem with the current join-point selection algorithm is when part of the image is just a region of black. This occurs when images have been joined and black is used to fill the area of the image that falls outside the bounds of either of the two images. After a pair of images has been joined, any rows or columns of purely black pixels in an image are removed from the images edge. This helps prevent compounding the problem in each stage of mosaicking, however does not remove it. When a segment is extracted from an image, it may contain an area of black. This is undesirable as there are no feature points that can be identified within the black area. To solve this problem, the segment extraction stage of the join-point selection should adjust the position of the segment if it contains a large area of black pixels at its edge.

When mosaicking an entire painting, discarding sets of join-points depending on the orientation difference does not work as intended. As sections of the painting get larger, a small angle can produces a large displacement at either end of the painting. Instead of using the angle between the images, the difference between the displacements at each

end of the image could be used. This measure would be independent of the size of the image.

One thing that has not been considered by the current join-point algorithm is the three dimensional nature of the x-rays. A possible approach to solving this problem for the case of nails in the images is to identify areas of the image containing a nail by thresholding the images to be joined. This step would only be necessary for images at the edge of the mosaic as they are the only ones to contain nails. This is because the nails are used to attach the canvas of the painting to the wooden support. By carefully selecting a threshold value it may be possible to identify the nails within the image as they are often much brighter than the rest of the image. Once the areas of the image containing the nails have been identified, the edge data for around those areas can be set to zero. This would be done before stage 2 of the join-point algorithm, removing them from consideration as points of interest in the rest of the algorithm.

4.2 Possibilities for Algorithm Optimization

It is possible to optimize the join-point selection algorithm in a couple of ways. These optimizations were not implemented in order to keep the code as simple as possible, making development of the algorithm easier.

The first point where optimization is possible is in stage 2 where the maximum points are selected for extraction into regions. On the first iteration of the algorithm, the 16 maximum points are located for extraction. If the algorithm fails on its first attempt to find a set of join-points, the same piece of code to find the maximum points is run, only the number of maximums that are kept track of increases. This could be optimized to find the top 128 points (the number of points used in the final iteration), and on each iteration use an increasing amount of the points. This would remove the need to scan the image looking for maximum points on every iteration; it would only be necessary on the first. Implementing this may not be worthwhile as the amount of time taken to run the relevant piece of code is insignificant compared to the runtime of the entire algorithm.

There is the possibility to increase the speed of the algorithm significantly in the case where the first attempt to find a set of join-points fails. The part of the algorithm that takes the most amount of time to run is in stage 3, where regions are matched to their opposing segments. This stage involves performing numerous pixel value subtractions and is a significant bottleneck. On the second iteration of the algorithm, 32 regions are extracted from each segment and matched to the opposing segments. These 32 regions include the 16 regions that were used in the first iteration. Instead of matching these 16 regions to the opposing segment a second time, the data should be saved from the previous iteration and only the new 16 regions should be matched. This idea should be extended to all iterations.

References

1. http://www.artmuseums.harvard.edu/mondrian/glossary.html accessed on 12/05/2005

2. W. Niblack, “An Introduction to Digital Image Processing”, Prentice Hall, London, 1986.

3. B. Zitová and J. Flusser, “Image registration methods: a survey”, Image and Vision Computing, Vol. 21, pp. 977-1000, 2003.

4. J. Hsieh et al., “Image Registration Using a New Edge-Based Approach”, Computer Vision and Image Understanding, Vol. 67 No. 2, pp112-130, 1997.

5. N. Chiba and H. Kano, “Feature-Based Image Mosaicing”, Systems and Computers in Japan Vol. 31 No. 7, pp. 1-9, 2000.

6. J. Hsieh, “Fast stitching algorithm for moving object detection and mosaic construction”, Image and Vision Computing, Vol. 22 No. 4, pp. 291-306, 2004.

7. W. Li and H. Leung, “A Maximum Likelihood Approach for Image Registration Using Control Point And Intensity”, IEEE Transactions on Image Processing, Vol. 13 No. 8, pp. 1115-1127, 2004.

8. http://www.vips.ecs.soton.ac.uk/ accessed on 12/05/2005.

9. http://www.vips.ecs.soton.ac.uk/nip2.php accessed on 12/05/2005

10. Hans Holbein the Younger, “Jean de Dinteville and Georges de Selve (‘The Ambassadors’)”, NG1314, The National Gallery, London, 1533.

Appendix A. Photograph of the Test Image

Appendix B. Source Code and Test Data (CD)