Image Alignment using Harris Corner Detection

Darragh Glavin: 16189183
Lorcan Williamson: 16160703
Yilin Mou: 18111602

1 Introduction

Image alignment is a common machine vision task, which involves combining two image into a single image. There are number of algorithms that can be used for this, falling into two categories, intensity-based, and feature-based. The algorithm that will be used in this project is feature-based, using the features common to both images to align them. To find these features, Harris corner detection was used.

2 Methodology

As mentioned the algorithm used to align the features that are common to both images. This algorithm can be broken down into 3 main steps;

1. **Feature detection:** For a feature to be useful for this task it must be unique within the image, i.e. features such as edges are not of much use for alignment. This meant finding regions of high local variance, such as corners. Harris corner detection is a good method for finding these points, and is what was used for this project.

2. **Feature matching:** The next step is to form pairs these Harris interest points found in each image. This is done by taking a patch around each interest point and creating pairs based of the most similar patches.

3. **Alignment:** The last step is to find the best translation to align the images. This is commonly done using a RANSAC search to find the an appropriate translation without checking all combinations of points. However, as there is a small (≈ 300) number of possible translations an exhaustive search can be done to find the 'best' translation.

Using these steps the sample image pairs were able to be aligned. However these were very clean images, and not very representative of a real world image alignment task. For this reason the effect small rotations and scaling of one of the images had on the alignment process was looked at.

3 Harris Corner Detection

The first step of the alignment process was finding the Harris interest points in each image. These are regions of the image that have a large variation in every direction. This is by thresholding and non-max suppressing the Harris response of the image. A grayscale version of the image was used to calculate the Harris interest points.

Harris Response

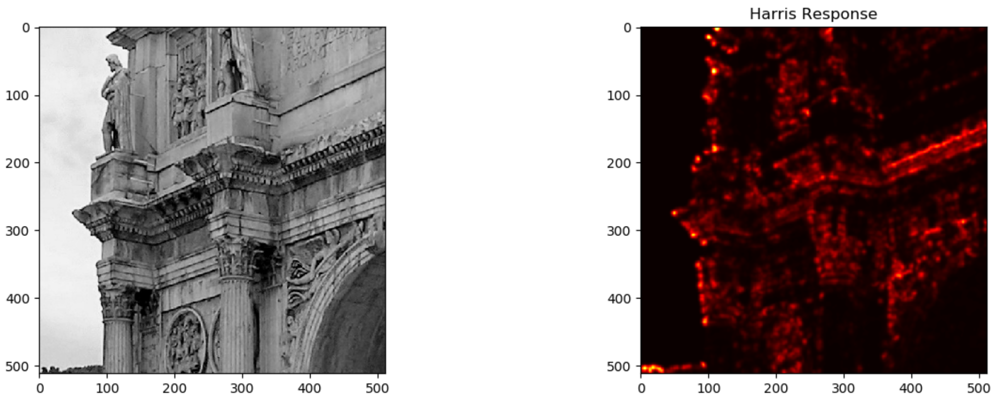
The Harris response of an image is equivalent to the smallest eigenvalue of that images *structure tensor*, also known as the second-moment matrix of the image. This is calculate using the partial derivatives of the image in the x and y direction. These are equivalent to detecting the edges in the x and y directions, and so they are calculated as the convolution of the image with a the partial derivative of the Gaussian filter in both x and y. The structure tensor of the image can then be calculated as shown in in equation 1.

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \tag{1}$$

Computing the eigenvalue of this would be costly however, so it is instead approximated by the Szeliski harmonic mean (Eqn. 2).

$$R = \frac{det(M)}{tr(M)} = \frac{AC - B^2}{A + C} \tag{2}$$

This gives the Harris response for the image, as seen in figure 1. Note how the sky exhibits no Harris response as it has almost no variation.



(a) Grayscale image

(b) Harris Response

Figure 1: Grayscale version of the image, and its Harris response

Harris Interest Points

However we cannot use the Harris response directly to align the images. As previously mentioned we must instead match patches from around the Harris interest points in each image. These interest points are considered as the largest Harris response values. We therefore threshold the Harris response to get the points greater than some percentage

of the maximum response value. For this project a threshold of $0.1 \times R_{max}$ was found to give good results.
The patches from around these points must be unique and not overlap. This means we must then non-max suppress the thresholded response. This removes Harris interest points that lie within the descriptor patch surrounding a larger valued interest point. Interest points around the edges of the images were also removed. A patch descriptor size of 11×11 worked well for the test images. After thresholding and non-max suppression the Harris interest points seen in figure 2 were left.

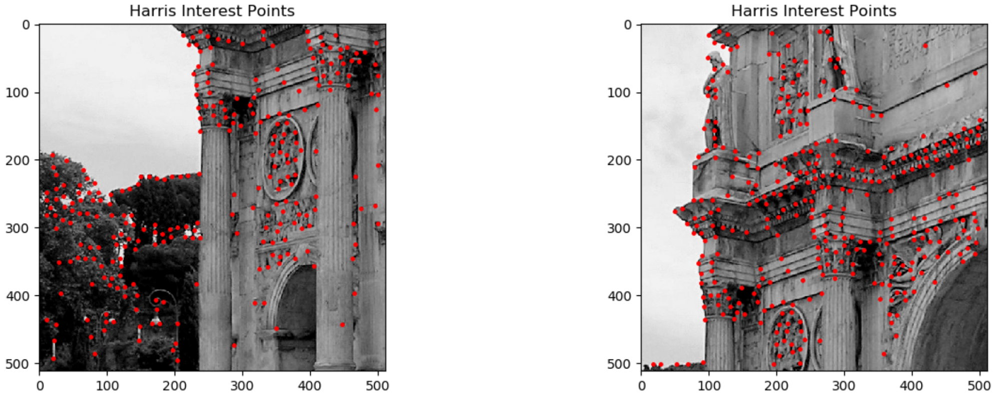


Figure 2: Harris interest points for one pair of test images

4 Feature Matching

Now that we have the Harris interest points of the images, we can begin creating pairs of matching features between them. The first step of this is to build up sets of patch descriptors. Each patch descriptor is a flattened view of the 11×11 area around each Harris point. These vectors are then normalised, and stacked into a matrix. This gives two $N \times 121$ matrices M_1, M_2 for each image, where N is the number of Harris interest points in the image (Fig 3).

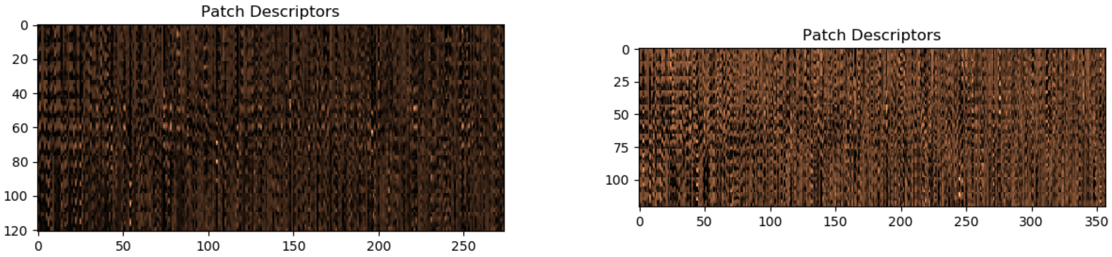
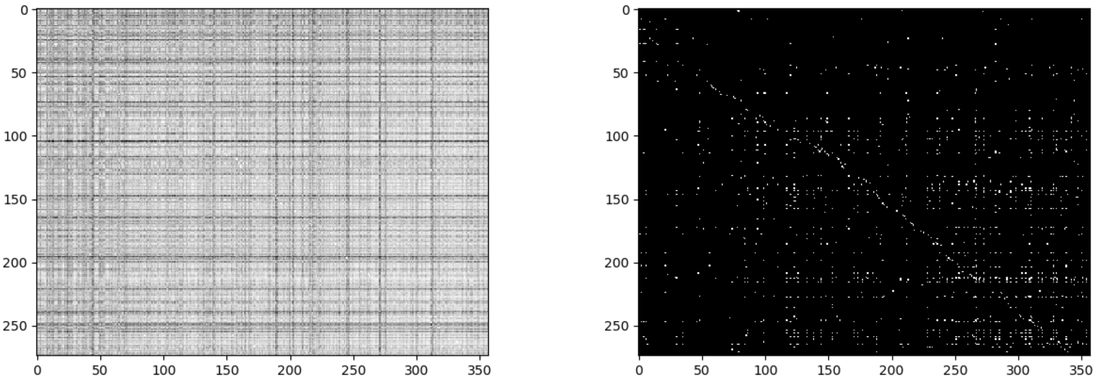


Figure 3: Patch descriptors of the two images (Transposed for clearer viewing)

Taking the dot product of the these two matrices $R_{12} = M_1 \cdot M_2^T$ shows how similar each patch descriptor in image 1 is to each patch descriptor in image 2. The values in R_{12} will be in the range $\{0, ..., 1\}$, where 0 means the patch descriptor vectors are perpendicular (not similar) and 1 means they are parallel (identical). R_{12} is then thresholded at a level of 0.95. This leaves a matrix where the non-zero value co-ordinates represent the pairs of matching Harris interest points between the two images (Fig. 4).



(a) R_{12}

(b) R_{12}, thresholded at 0.95

Figure 4: $R_{12} = M_1 \cdot M_2^T$ before and after thresholding

5 Alignment

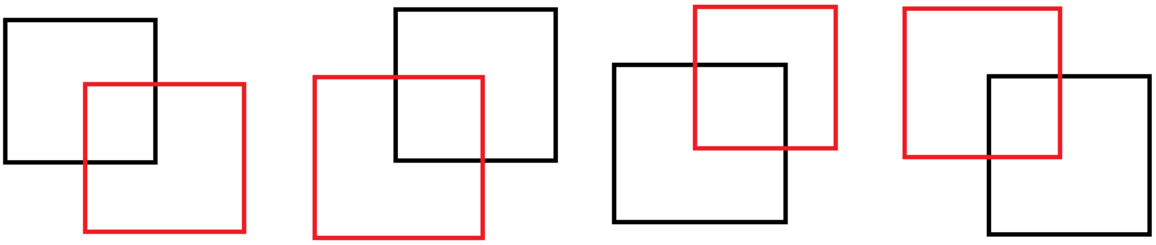
Using these pairs of matching interest points a set of translation vectors can be made. These represent how image 2 would need to be translated to line the pair of interest points. However these do not all agree, as some matching pairs may be false matches. This means some 'optimal' translation vector must be chosen from the set of possible translations. This optimal vector would be one that the majority of translations agree with. This would normally be done using a RANSAC (RANdom SAmple Consensus) algorithm, which uses random sampling to estimate some property of a set that may contain outliers. Random sampling is used to remove the need for an exhaustive search which is computationally expensive for large sets. However in this case the set of possible translations is relatively small (≈ 300), so an exhaustive search for the translation with the highest consensus can be done.

For this the difference between some test translation and all the other possible translations was calculated. Then these differences were thresholded, and if the difference was less then a threshold T the test translation received a vote from that translation. The is was done for all possible translations, and the test vector with the most votes was considered the optimal translation (Eqn. 3). A threshold of $T = 1.6$ pixels was found to work well.

$$f(\vec{v}, \boldsymbol{V}) = \sum_{\vec{x} \in \boldsymbol{V}} \begin{cases} 1, & |\vec{v} - \vec{x}| \leq T \\ 0, & |\vec{v} - \vec{x}| > T \end{cases}$$

$$\vec{v}_{optimal} = \arg\max_{\vec{v}}(f(\vec{v}, \boldsymbol{V})), \ \forall \ \vec{v} \in \boldsymbol{V} \tag{3}$$

Once this translation is found there are 4 possible ways image 2 could be placed relative to image 1, as shown in figure 5.



(a) $(+\Delta x, +\Delta y)$ (b) $(-\Delta x, +\Delta y)$ (c) $(+\Delta x, -\Delta y)$ (d) $(-\Delta x, -\Delta y)$

Figure 5: Possible arrangements of the two images.

A blank image is made into which the two images can be placed. The 4 possible translation vector cases gives 4 sets of locations to place the upper left hand corner of each image within the final image, and 4 sets of size for this final image ((x, y) co-ordinates used);

Translation vector	$\boldsymbol{I_1}$ **origin**	$\boldsymbol{I_2}$ **origin**	**Combined image size**
$\boldsymbol{(+\Delta x, +\Delta y)}$	$(0, 0)$	$(\Delta x, \Delta y)$	$(x_2 + \Delta x, y_2 + \Delta y)$
$\boldsymbol{(-\Delta x, +\Delta y)}$	$(\lvert\Delta x\rvert, 0)$	$(0, \Delta y)$	$(x_2 + \Delta x, y_1 + \lvert\Delta y\rvert)$
$\boldsymbol{(+\Delta x, -\Delta y)}$	$0, \lvert\Delta y\rvert)$	$(\Delta x, 0)$	$(x_1 + \lvert\Delta x\rvert, y_2 + \Delta y)$
$\boldsymbol{(-\Delta x, -\Delta y)}$	$(\lvert\Delta x\rvert, \lvert\Delta y\rvert)$	$(0, 0)$	$(x_1 + \lvert\Delta x\rvert, y_1 + \lvert\Delta y\rvert)$

This allows the two images to be aligned as seen in figure 6. For the test images used, perfect alignment could be achieved for all pairs.

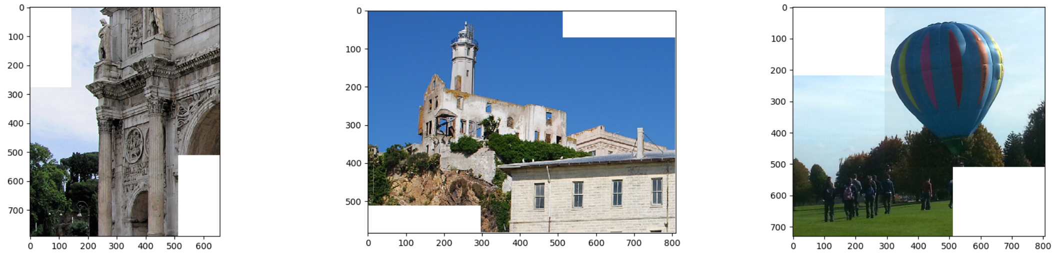


Figure 6: Aligned test images.

6 Effect of Transformations

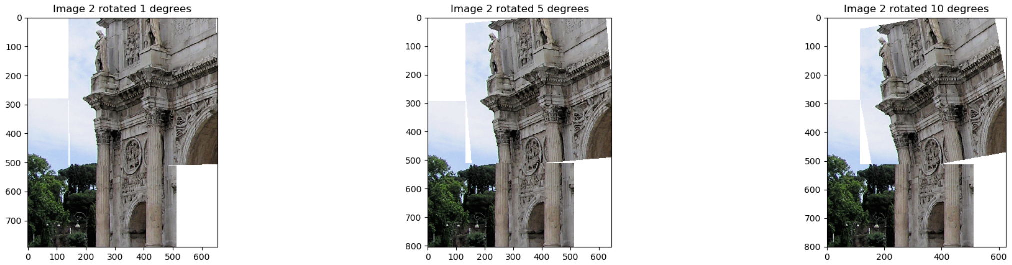
As mentioned the test images used were very clean, and not representative of real world images that may need to be stitched, such as when creating panorama shots. This section therefore will look at how transformations, in this case small rotations (±10°) and scaling (50%–200%) of one of the images, effect the algorithms ability to align the images.

Rotations

The first transformation tested was rotating one of the images relative to the other. Angles between 1° and 10° were tested. Rotating the images introduced larger regions of 0 value pixels into the images. This cause undefined NaN pixel values after division in 2. To avoid this equation 3 was changed into 4, where ϵ is some small value, in this case 1×10^{-10}.

$$R = \frac{det(M)}{tr(M) + \epsilon} = \frac{AC - B^2}{A + C + \epsilon} \tag{4}$$

Using this new formula the alignment algorithm could be used on the rotated images, and the results can be seen below in figure 7. The two images are no longer aligned, as expected, however they are still placed in roughly the correct spot relative to each other.



(a) 1°

(b) 5°

(c) 10°

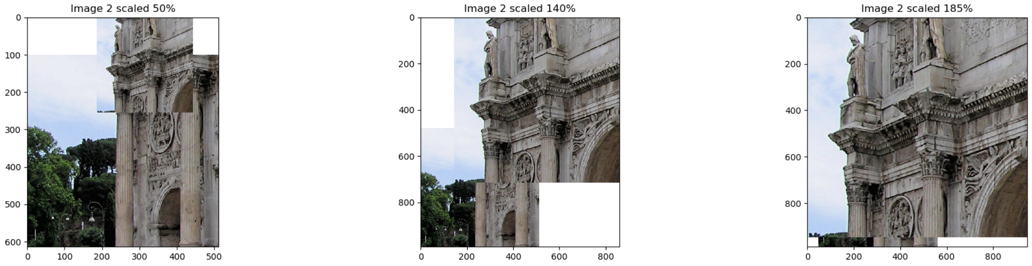
Figure 7: Images aligned after rotating.

Scaling

The other transformation looked at is the effect of scaling one of the images. For this one of the images was scaled to between 50%–200% of its original size. For this steps of 15% were used. This also introduces an error due to an assumption made when calculating the combined image size, that one image edge would not be overlapped after translation by another. To resolve this the size formula had to be modified slightly for each possible translation:

Translation vector	**Combined image size**
$(+\boldsymbol{\Delta x}, +\boldsymbol{\Delta y})$	$(max(x_1, x_2 + \Delta x), max(y_1, y_2 + \Delta y))$
$(-\boldsymbol{\Delta x}, +\boldsymbol{\Delta y})$	$(max(x_1, x_2 + \Delta x), max(y_1 + \lvert\Delta y\rvert, y_2))$
$(+\boldsymbol{\Delta x}, -\boldsymbol{\Delta y})$	$(max(x_1 + \lvert\Delta x\rvert, x_2), max(y_1, y_2 + \Delta y))$
$(-\boldsymbol{\Delta x}, -\boldsymbol{\Delta y})$	$(max(x_1 + \lvert\Delta x\rvert, x_2), max(y_1 + \lvert\Delta y\rvert, y_2))$

Figure 8 shows the effect of rescaling on the alignment. From these figures it can be seen that the relative positions are still correct when aligned, but that as expected the images don't fit together.



(a) 50% scaling

(b) 140% scaling

(c) 185% scaling

Figure 8: Images aligned after scaling.

7 Results and Conclusions

This project shows image alignment can be done using feature matching through Harris corner detection. It was seen that for the test images used, this method was able to achieve perfect alignment. However, it was also seen that if there is any difference in orientation or scaling between the two images, then the alignment will succeed. The approximate location of the two image relative to each other will be correct, however as expect they do not fit together. For them to align perfectly some extra step would need to be added to the algorithm to calculate a transform that would orientate and scale the images so they could fit together.