Abstract

This project experiments with algorithms which could address problems in Time – lapse photography, where image processing for an image sequence might be required to obtain better visual effects. Specifically, image registration is essential when images are not aligned decently, which could result in scene flickering when images are projected into films at play back stage, a motion – estimation based algorithm is applied to align images after searching for the best reference image. In addition, when a large motion is presented between two frames, a new frame could be created and this indicates adjusting the play back timescale to be non - linear, as a consequence, the movements become relatively slower and more details are shown to the audiences.

The methods are first experimented with two images and then extended to an image sequence, to put another way, when an image sequence is input to the program, it is completely automatic to detect where to apply registration and which image is selected as the reference. Similarly, interpolation point is auto – sensed as well. As a result, it could save time for manual operations on image sequence for time – lapse photographers.

In this paper, the algorithms that are employed are mainly explained in Section 3 after some basic concepts in Section 2, a list of bullet points are displayed as follows:

- Implementation of threshold – XOR method (proposed by G. Ward, 2003) for image registration (Section 3.1.1).
- Motion estimation method for image registration and image interpolation (Section 3.1.2 and Section 3.2.1).
- Detection of reference image for image sequence registration (Section 3.3.1).
- Finding an interpolation point in image sequence (Section 3.3.2).

Contents

1. Introduction

1.1 Non – Linear Time – Lapse Photography

Time – lapse photography, by definition, is a process containing long – time image sequence capturing and then playing back at some rate (usually 24 frames per second), as a piece of animation, for example, a sunrise and sunset procedure can be recorded by digital camera every five minutes during an entire day, and software can be used for the short film construction.

Since the capturing stage covers a long time, a tripod is often used to fix the camera in the same position, however, if capture work lasts for weeks, it could be extremely difficult to keep the tripod in the exactly same position, therefore, image registration is required to avoid flickering when they are playing back.

Time appears to be moving fast and thus lapsing when the image sequence is played at a certain frequency, which is often 24 frames per second, to put another way, 24 images are played in sequence on screen within a second. If such frequency is modified, from the audiences' perspective, objects in the scene would move faster (frequency increased) or slower (reduced), which, indicates a non – linear playing back rate.

In this project, non – linear time is implemented by image interpolation techniques, if new images containing intermediate positions between two neighboring frames, it would be tantamount to lessen the frame rate, for a smaller number of images will be presented per second and thus, the speed of moving parts in the scene would be felt slower.

1.2 Aims and Objectives

This project explores algorithms that address problems in capture and playback phases of time – lapse photography. When capturing images, a tripod is indispensable to fix the position of a camera, however, usually it could be massively time – consuming for this process which could cover longer than weeks, thus a practical problem is that it is difficult to keep the camera exactly in the same position, and the first aim of this project is to find a proper method to implement registration for image sequence, including an algorithm to align two images as well as deciding the best image to act as the reference.

- Objectives for image registration

- Implementation of an algorithm to align two images.
- Apply the algorithm to image sequence.
- Experiment with the algorithm for acceleration.

In the playback stage, the main aim is the non – linear time, on the one hand, to increase the speed, removal could be easily applied in an certain software, thus the project primarily focus on the opposite aspect – to reduce the playing back frequency. Image interpolation is utilized to achieve this aim.

- Objectives for image interpolation
  - Implement an algorithm which is able to create a new image between two neighboring frames.
  - Apply the algorithm to image sequence.
  - Experiment with the algorithm to enhance the plausibility of the new frame.
  - Find methods to automatically decide where to create new frames.

1.3 Structure

This report consists of five main sections, firstly, background of relative methods and techniques in conjunction of previous research, are described in the following section, where a number of registration and interpolation methods are included. In section 3, there are methods designed for this project, including those are unsuccessful after experiments. Then the implementation and results section mainly explains thresholds selection and parameter configuration for relatively better segmentation, and thus obtaining a plausible result. The next section is a discussion of the results from experiments, followed by an evaluation of the project to determine if it has obtained the objectives. Finally future work is discussed in different aspects, including software developing and extensions of present methods.

2. Background

2.1 Image Segmentation Methods

In this part I will describe two fundamental image segmentation techniques that are relative to this project, one is to divide an image into two parts by a threshold and the other is based on sub – divisions.

2.1.1 Thresholding

Thresholding requires gray – level images and is the relatively simpler method segmentation process, it could work well when an image owns a decent contrast, which is suitable for a constant threshold to distinguish the background and objects in an image. A basic thresholding algorithm can be described as follows:

Algorithm 1 Basic Thresholding

Search all pixels f(x, y) of the image f. An image element g(x, y) of the segmented image will be categorized as object if f(x, y) > T, where T denotes the threshold, otherwise, it belongs to the background.

There are a number of methods to determine T, usually detecting the mean or median gray value could be suitable for segmentation with brightness. Mean gray value is calculated from the mean value of all pixels, while median is often the gray value of the pixel at the median position after sorting all pixels.

2.1.2 Region – Based Segmentation

A natural region – based method for segmentation is region growing. Beginning from the raw image data after sub – division, two adjacent regions will be merged if they satisfy some merging criterion, then try the next region until no pair matches the condition. On the opposite, segmentation could also be implemented by splitting to small regions from the entire image, furthermore, splitting and merge techniques could be used simultaneously.

When merging and splitting are combined, it works in a hierarchical data structure (M. Sonka, et al, 1999), images are sufficiently subdivided when not matching a homogeneity criterion, then merge adjacent regions if they could be in a homogeneous region.

2.1.3 Summary

Thresholding and region – based methods are basic but significant techniques for image segmentation, in this project, segmentation is utilized before processing images and some thoughts are adopted from these fundamental algorithms.

2.2 Optical Flow and Motion Estimation

2.2.1 Optical Flow

Optical flow indicates the motions in images during a small time interval, including movements from objects, edges, surfaces and even the entire scene (camera movement). Basic optical flow estimation is on the basis of the Optical Flow Equation (OFE)

$$\frac{dE}{dx} V_x + \frac{dE}{dy} V_y + \frac{dE}{dt} = 0 \tag{1}$$

Where $E(x, y, t)$ is the image sequence, V_x, V_y denote the 2 – D velocity in the image plane respectively, while $\frac{dE}{dx}, \frac{dE}{dy}, \frac{dE}{dt}$ are partial derivates of intensity within a frame and between frames. Equation (1) indicates that intensity along motion trajectory remains constant.

2.2.2 Motion Estimation in H.264 / AVC Introduction

H. 264 / AVC is a commonly used compression, recording and distribution standard for high definition video, and motion estimation techniques form the core of such applications (Iain Richardson, 2010). Source images could be captured from either digital cameras or a video clip, a motion vector (x, y) is used for representing motion information from one frame to a reference frame, and an image could be sub – divided into small motion blocks (usually square - shaped), each motion block contains a motion vector when compare to the reference.

H.264 / AVC differs from other MPEG encoding standards mainly during motion estimation together with its two components, motion vectors and motion compensation. When operating motion estimation, image information is computed for similarities that can be reused in subsequent frames. This ultimately reduces the volume of data that is encoded and therefore reduces the bit rate.

Block Matching Algorithm (BMA) (I.E.G Richardson, 2003) is a popular method in motion estimation, as shown in Figure 1, the algorithm is described as follows:

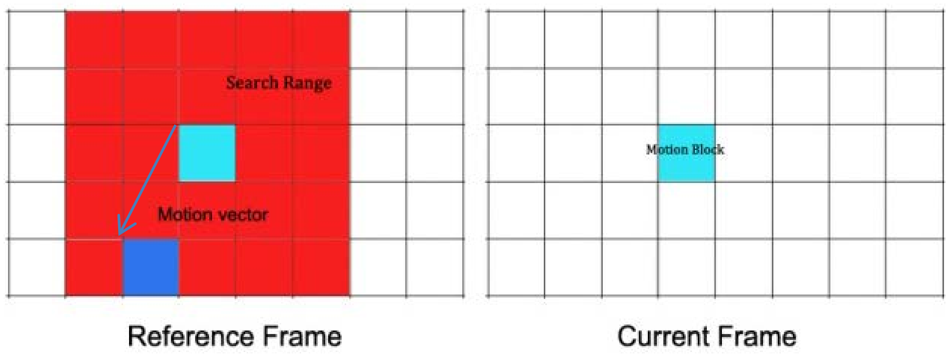


Figure 1 Basic concept of BMA

Algorithm 2 Block Matching

1) For one motion block in the current frame, we compare it within the search range (a larger region containing same size blocks around the corresponding block) in the reference frame, for each comparison.
2) A motion vector is recorded along with some figures that denote similarity.
3) Finally, the motion vector with the largest similarity will be selected, and all pixels in the block are regarded as having the same motion vector.
4) Repeat from 1) until all the motion blocks are checked in the current frame.

2.2.3 Block Matching Algorithms

In this part a number of BMAs are described and compared, basic algorithms includes full search and relatively limited search are demonstrated with figures, they are compared mainly in the amount of computations, it is predictable that computation and estimation accuracy are in positive relationship, lager amount of computation obtains more accurate estimation results.

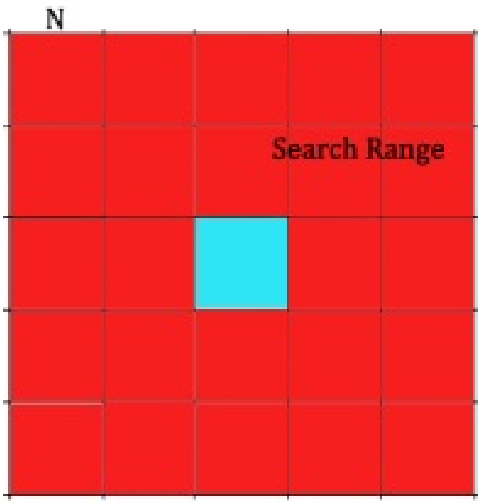


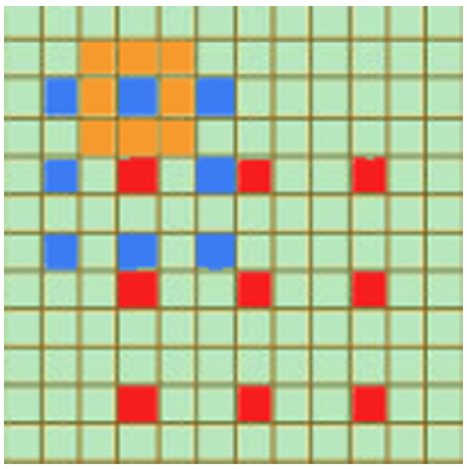
Figure 2 Example of a definition of motion block and search range

Block size: N * N Pixels

Full search block matching

Full search block matching algorithm (Alois, 2009) assesses every possible motion blocks in the search range. Hence, it could generate the best block matching motion vector. Residue is less possible in this type of BMA, however, the required computations are relatively high due to a larger amount of candidates to evaluate in a pre - defined search range. If a motion block is defined as $N * N$ pixels with search range of $\pm 2N$ disrance. A full search BMA evaluates every blocks, in the search range at one pixel step. Hence, as shown in Figure 2, the number of candidates to evaluate is $4N * 4N = 16N^2$, which is predominantly high compared to any of the search algorithms.

There are several other faster block-matching algorithms, which could reduce the number of evaluated candidates yet try to keep good block matching (Yu-Wen Huang, et al, 2006) accuracy.



🟥 First iteration

🟦 Second iteration

🟧 Third iteration

Figure 3 Illustration of Three – step

Three step search

In a three-step search (TSS) algorithm (Alan Bovik, 2009), whose process could be described as Figure 3, the first iteration evaluates nine candidates, in a relatively lager step. The candidates are centred on the current block's position. The step size for the first iteration could be set to half the search range. Then, for example, the candidate on the top left could be the most probable motion of the block, thus the next iteration centres on that block and search for eight surroundings again, but with step of one block closer, at this stage, another motion vector is found and the algorithm enters the third iteration, assessing the adjacent eight blocks, finally obtains the motion information. To sum up, TSS compares 9 blocks during each iteration (the centre position included), in total the computation is less than full search block matching.

2D logarithmic search

Similar to TSS algorithm, 2D Logarithmic (Alois, 2009) search is another algorithm, which tests limited candidates in neighbourhoods. As displayed in Figure 4, during the first iteration, five candidates are tested. The candidates are centred on the current block location in a diamond shape. The step size for first iteration could be set equal to half the search range. In the second iteration, the centre of the diamond is shifted to the best matching block. Only if the best match block happens to be the centre block, the step size will be halved, otherwise, if the best candidate is not the diamond centre, step size remains the same. In this case, some of the diamond candidates are already evaluated during first iteration and they are not considered during the second iteration. The results from the first iteration can be used for these candidates. When the step size is reduced to only one pixel, it begins the final iteration where all eight surrounding candidates are evaluated. The best matching candidate from this iteration is selected for the current block.

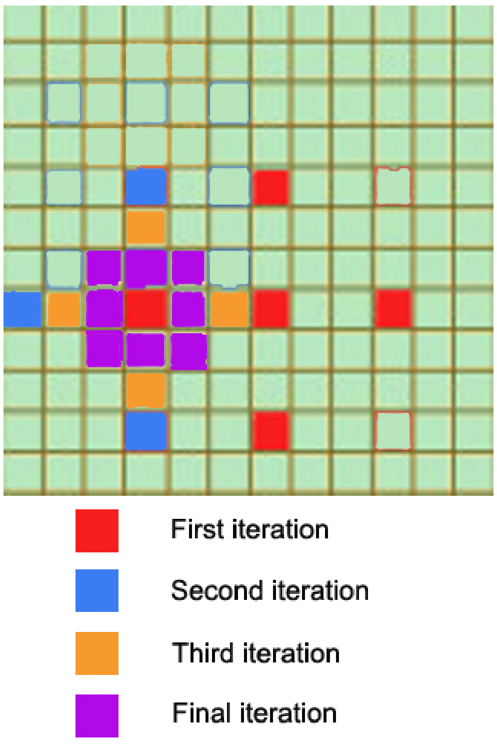


Figure 4 Illustration of 2D logarithmic search

One at a time search algorithm

The one at a time search algorithm estimates the motion vector independently in x and y directions. Firstly the algorithm computes blocks in horizontal direction. During each iteration, three adjacent two blocks along the x - axis are tested as shown in Figure 5. The three blocks are together shifted towards the best matching candidate, with the best matching block in the centre. The process terminates if the best matching candidate happens to be the centre of the candidate set. The block position relative to the original in this direction is used as the x - component of the motion vector. Then the same process is applied vertically, in the y - axis. It is similar to x-axis search and is followed by estimating the y - component of the motion vector. One - step at a time search on average tests less number of candidates. However, the motion vector accuracy is poor.

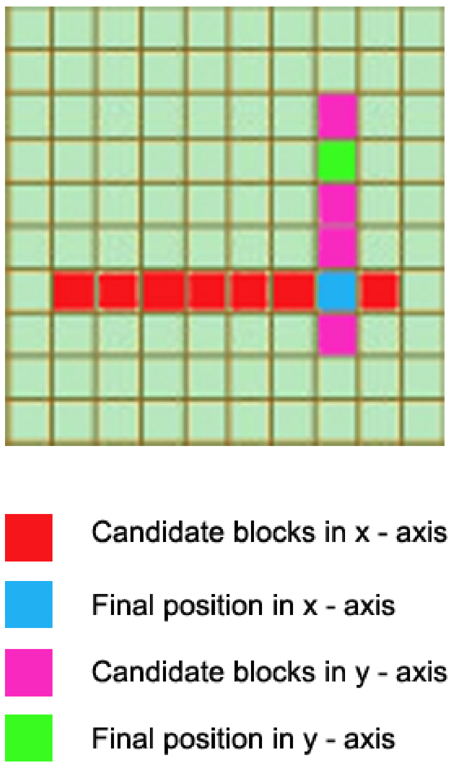

🟥 Candidate blocks in x - axis
🟦 Final position in x - axis
🟪 Candidate blocks in y - axis
🟩 Final position in y - axis

Figure 5 Illustration of One at a time search

2.2.4 Difference Criteria

For a certain motion block, the purpose of motion estimation is to detect a 'matched' block somewhere in the context. There are a variety of methods to measure differences between two motion blocks, and the selection of these measurements depends on the practical work, actually it is difficult to establish a perfect criterion, in this part two commonly used standards are described in detail.

Means squared difference

The Mean Squared Difference (MSD) is firstly calculate the difference of pixel values between the current frame and the reference frame at pixel p, then square the difference and finally add up the results from all the pixels in one comparison. Specifically, for a block containing $N * N$ pixels, the MSD could be expressed by the following equation.

$$MSD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}[f_{cur}(i,j) - f_{ref}(i,j)]^2$$

(2)

Where $f(i,j)$ denotes the pixel value at position (i,j),and multiply by $\frac{1}{N^2}$ is for the purpose of the mean value, in practice, it could be neglected if magnitude makes contributes to decision making, where smaller values indicates a better match, and hence the division operation makes no difference for the result.

Mean absolute difference

A similar criterion to MSD is the Mean Absolute Difference (MAD), which only takes the difference of pixel values rather than squaring it, it is shown by the following expression.

$$MAD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|f_{cur}(i,j) - f_{ref}(i,j)|$$

(3)

Comparison

There are no significant differences between MSD and MAD, in both criteria, larger similarity is expressed by smaller values, however, from a graphic demonstration in Figure 6, it can be seen that the MSD provides more smoothness in a 2 – D coordination system.

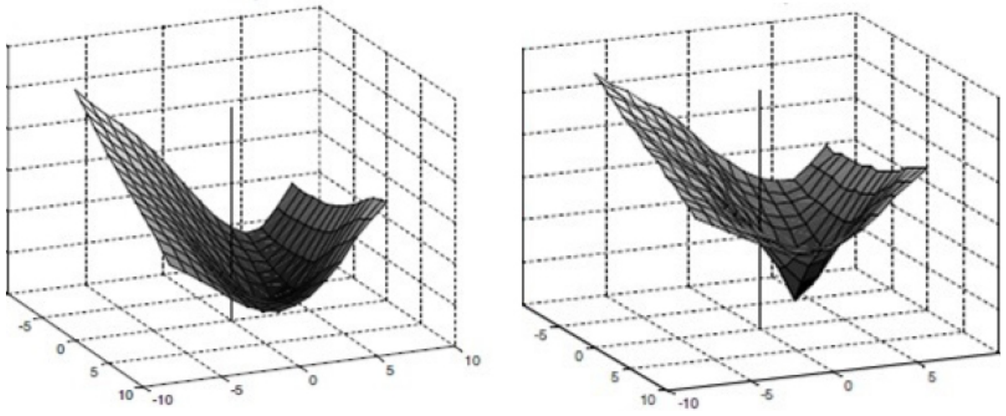


Figure 6 Maps of the MSD (left) and MAD (right)

(Murali, et al, 2012)

2.2.5 Summary

As researching with motion problems, the basic concept is the principle of optical flow – constant intensity along motion trajectory, so that pixels in the newly created frames could be copied from input frames and thus benefits consistency. The BMA in motion estimation could implement segmentation to distinguish the background and the moving objects between frames, there are a number of methods, some emphasize speed while they may lost accuracy, therefore, accuracy grows with computation, it is significant to consider both factors, depends on the particular work at hand.

2.3 Image Registration

2.3.1 Introduction

Image registration is the process of transformation of coordination systems, from different ones into one. In this project, registration mainly is represented by image alignment operations, which could avoid flickering when the image sequence is played back, it is significant since the photographer cannot ensure that the tripod are installed at exactly the same position throughout the entire capturing phase, which could last longer than weeks or even months. In addition, as I experimented in practice, even if the tripod position are carefully marked, it is difficult to capture the a consistent sequence after moving away from original place, thus, registration is indispensable for generating a more reasonable sequence.

Another factor in time – lapse photography that requires consideration is the registration should only be applied to the background rather than the motion parts, which should actually be intentionally neglected when implementing the registration algorithm, therefore, segmentation is essential before alignment.

2.3.2 Previous Work

Image registration has been applied to different regions, for example, virtual reality (A. Aseem, et al, 2006), analysis of medical images (K. Loewke, et al, 2008) or for military purposes (N. Heinze, et al, 2008). There are also a number of methods to implement alignment process, the method introduced in (Q. Zhang, et al, 2011) is a popular feature matching algorithm, with the combination of Fourier Transform and SIFT feature point extraction for efficiency and robustness purposes, while when applied to time – lapse photography, features from motion objects might also be detected and thus affects the results. (F. L. Seixas, et al 2008) proposed an Genetic Algorithm (GA) – image registration method, creating a new perspective for matching and mapping point sets, but it is difficult to learn images for such a substantial quantity.

An automatic and robust algorithm introduced (Q. Zhang, et al, 2011) which is implemented by matching features of two images, Fourier Transform (FT) is employed for relative points calculation and Scale-invariant Feature Transform (SIFT) can be decent algorithm boils down to retaining features like illumination, rotation, or geometric distortion. Their experiments have shown that it is less expensive in extracting feature points to combine FT and SIFT. In FT, image transformation pair is assumed as *f(x, y)* and *F(ξ, η)*, with displacement *(Δx, Δy)*, thus the transformation can be expressed as:

$$f\ (x\ - \Delta x, y\ - \Delta y) \Leftrightarrow F(\xi,\eta)e^{-j2\pi(\xi\Delta x+\eta\Delta y)} \tag{4}$$

For two images, $f_1(x, y)$ and $f_2(x, y)$, their relationship in time domain can be assumed as

$$f_1(x,y) = f_2\ (x\ - \Delta x, y\ - \Delta y) \tag{5}$$

Therefore, FT of each image can be obtained from (1) and (2):

$$F_1(\xi,\eta) = F_2(\xi,\eta)e^{-j2\pi(\xi\Delta x+\eta\Delta y)} \quad (6)$$

At this stage, phase correlation function can be calculated by cross power spectrum of inverse FT:

$$hql(x,y) = \delta\,(x - \Delta x, y - \Delta y) \quad (7)$$

where *(Δx, Δy)* is a non - zero position of phase correlation function which values in zero at most positions, and it is the shift that expected for image overlap region segmentation.

The algorithm preferred at early stage is (G. Ward, 2003), which is original designed for High Dynamic Range (HDR) images composition, during which capturing of several source images is similar time – lapse photography. This method thresholds images after transforming into grayscale color space, then applies an exclusive – or (XOR) operator to show misaligned coordinates, before adjusting x and y offsets. This method neglects the variations in intensity in different images, while it could be true in time lapse photography, it is implemented in this project and results are discussed in section 4.

2.3.3 Summary

In conclusion, as it is time – consuming for capturing images, and from practical experiments, registration is a first step in addressing problems in such image sequence, specifically, as a compensation for slightly movements of the camera to ensure a smoothly changes between frames, a number of algorithms could be adopted and then applied to the entire sequence.

2.4 Image Interpolation

2.4.1 Introduction

Current films are usually projected at 24 frames per second, which means 24 images appear in each second (linear time), an interesting challenge in this project is to adjust the playing back speed when necessary.

Image interpolation is the method that is employed in this project to implement a non – linear timescale, by definition, image interpolation can also be regarded as image scaling and resampling, which will increase the image size and resampling to a newly – created image plane. Particularly, in time – lapse, the resampled image need a motion prediction algorithm, which could create a new image that depicting the scene time – related but not captured by a digital camera, but rescaling is not included in the scope.

Implementation of non – linear time results in a more expressive film because when something actually occurs between frames, the story could be displayed more in detail to the audience when new frames are created, objects appear to move slower than before while on the opposite, the process could be accelerated when nothing moves or only slightly movements take place.

2.4.2 Previous Work

A path – based algorithm is introduced in (D. Mahajan, et al, 2009), they constructed a path between two images where new images need to be inserted, and along this path, pixel gradients could be easily copied and moved to implement the interpolation, and the frequency contents of intermediate images could be reserved without ghosting and blurring, with temporal coherence maintained. (T. Stich, et al, 2011) proposed a method attempting to adapt human visual system for such image sequence, which included the indispensable motion prediction process. The two methods seem to perform in their own characters and I will describe them in detail respectively.

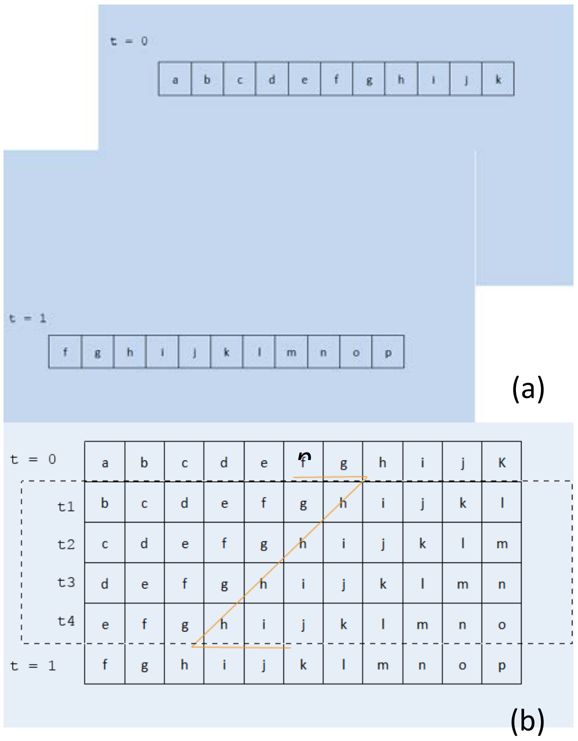


Figure 7 Basic notion of path

The basic idea of (D. Mahajan, et al, 2009) is to create some ‘motion path’ to lead the variation of pixels between two input images, the basic notion of such path is described in Figure 7, (a) defined two simple images constructed by single lines (blue background not included), and at t = 1 the image is a left – shifted version of the one at t = 0, they are together regarded as the input images in this example, Figure 7 (b) shows that for images could be interpolated between the inputs, and each time, the image is actually shifted to the left at a step of 1 pixel, character 'f' is the information on pixel p0, which will be interpolated at pixel p0 – 1, in the next image, so the algorithm copies 'f', from the input t = 0 image, and then copies 'g', 'h', respectively, after that, the copy operation will be moved to t1 image, which is indicated by the orange line in Figure 7(b), and this process comprise the path. In the t = 0 image, the p0 + 2 pixel (recording 'h') is the point that the path changes direction, this is defined as transition point, where the intensity values of two images match, and hence, the interpolation could be plausibly implemented.

Relative to interpolation, computing has been applied to optical flow as well as video segmentation (C. L. Zitnick, et al, 2005), but due to implicit occlusion reasoning, it is difficult to present some occlusions. Graph cuts are also employed for global optimization (V. Kolmogorov, et al, 2001) and (J. Sun, et al, 2005), however, they are difficult for explicitly occlusions considerations. With regard to image warping and morphing, an image – based rendering method is proposed by (A. W. Fitzgibbon, et al, 2003), which utilizes manual correspondence based on stero and optical flow. In addition, interpolation is also applied to video processing, (H. Wang, et al, 2004) proposed an algorithm for video editing based on 3D Poisson equation, while similarly in the gradient domain, with Poisson equation, there are applications for image editing(A. Agarwala, et al. 2004) and (J. Jia, et al, 2008). However these methods are not compatible to the interpolation in time – lapse photography because their inputs and outputs are both images or videos, which could be regarded as static methods.

2.4.3 Summary

The key factor for image interpolation is trying to create the most plausible images, from the previous work, it could be seen that the new frame data should be copy from the original ones, in which way the consistency in intensity and color could be kept to the largest extent, a robust algorithm for new position computation is the key element during this process, in addition, occlusion handling should be carried out as well for the completeness of the newly created frame.

2.5 Background Summary

This section has explained the reason for the registration and interpolation in this project, in conjunction of previous methods and algorithms that are relative to this project, from basic concepts to previous research work, algorithms that might be adopted are the project is described in detail.

3. Methods

In this section all the methods and algorithms that are employed in this project are described, they are primarily divided into two parts as per the operations of registration and interpolation. Firstly, the method from (G. Ward, 2003) is described in detail with modifications, then motion estimation based algorithm is found and applied to both operations.

3.1 Image Rgistration

This part contains methods for image registration. Firstly, the thresholding and XOR algorithm proposed by (G. Ward, 2003) is adopted and explained in 3.1.1, then, inspired by image interpolation, I experimented with motion estimation to align images, finally, in section 3.1.3, I describe the methods that benefits the processing speed, which is fairly significant in such image sequence solution.

3.1.1 Implementation of a previous Framework

Thresholding and XOR method overview

A fast and robust method proposed by G. Ward was taken into consideration in the early, the registration for HDR image construction might be relevant to some extent, since HDR images, similarly, requires several images to be overlapped with aligned perfectly before enhancement in color, intensity and contrast. A hardware solution is to set up a tripod to capture multi – images, however, this algorithm is designed for hand – held exposures, which could cause similar camera movement when creating time – lapse image sequence, hence, it should be worth experimenting with.

The input to the algorithm is two grayscale images, which could be converted by the following formula[1].

$$Y = 0.212671 * R + 0.715160 * G + 0.072169 * B \tag{8}$$

[1] This equation is from the OpenCV function that are used for color space transformation.

Then one of the two is selected as the reference, both images will be segmented by a median threshold, which is selected from sorting all the gray values, the segmentation algorithm is displayed below.

Algorithm 3 Median Thresholding

1) Compute grey values for all pixels in the image and then accumulating frequencies to form a histogram.
2) Begin from 0, count pixel numbers located in each grey value until they are add up to half of the pixel amount (image height multiplied by image width).
3) At this point, the value is selected as the threshold T.
4) Segment image, if pixel value $f(i,j) > T$, set pixel value as 255 (black), otherwise, set values to 0 (white).

This algorithm could segment image according to the exposure, brighter part will be completely set to white while darker part will be black, an

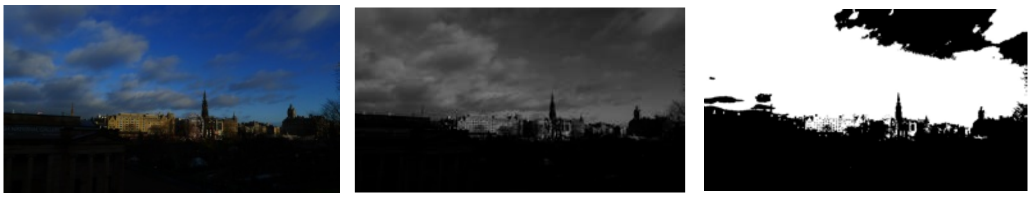


Figure 8 Results of grayscale and thresholding segmented images

Left: Original Image Middle: Greyscale image Right: Segmented image

example from images captured in Edinburgh, Scotland are shown in Figure 8 (with original image, grayscale image and segmented result), the result contains only two values and could be easy to implement the following XOR operation.

The identity of two images is measured by an exclusive – or (XOR) operator, which could get a value of 0 under the condition that two pixels own the same values, while non – zero if they are different (set to 255 for showing a recognizable result), an example of two unaligned images is shown in figure 9.

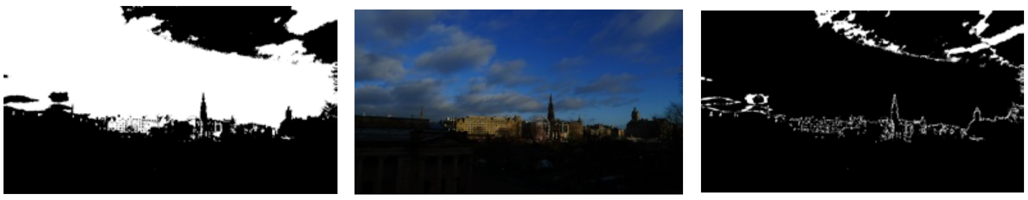


Figure 9 Result of XOR operation

In spite of the cloud in the sky, the outline of buildings indicates the difference, and it is noticeable that computing one image with itself will result in a completely black scene, thus to find the offsets for a pair, we select one as the reference, then apply XOR in four directions respectively, and calculate the total non – zero pixel quantity, the smallest direction will be regarded as the correct movement.

Acceleration

Figure 10 Image pyramid

When applied in practices, the basic method could encounter difficulties, naturally images from time – lapse photographer may have a High Definition (HD) resolution at 1920 * 1080, which could result in a time – consuming pixel comparison, or worse, a larger range to detect the displacement in contrast to smaller – sized images. A common solution is to create an image pyramid, which resizes images down by a quarter (halves height and width respectively), with such technique, computation cost will be massively reduced. Figure 10 gives a concept of image pyramid.

Now that the source images are shrunk to smaller sizes, we could limit a ± 1 pixel range for XOR comparison in horizontal (x) and vertical (y) directions respectively. Finally, if the image pyramid is created to L levels, returning consequence from the accelerated algorithm will be linearly expanded by a factor of 2 in each level, to put another way, the final shifting magnitude is 2^L times that of the bottom level.

In conclusion, to align images via threshold and XOR operation is less complicated comparing with others (SIFT, GA, etc.), the key factor is to compute differences in two images, in conjunction with acceleration method, the results are shown in section 4.2.1.

3.1.2 A Motion Estimation – Based Method

Due to the block matching algorithm could segment background and moving parts in an image, with motion information (motion vector) in each motion block, it is deducible that the motion vectors of the background parts could indicate the displacements between two images, simultaneously, for aligned images, that of moving objects could be a reference for interpolation, I will explain the registration in this section, while the interpolation algorithm will be given in section 3.2.

As described in section 2.2.2, if two images are input to an algorithm based on the BMA and one is selected as reference, blocks in the other image could find similarities by an search range in the reference, and thus obtaining a motion vector encoding as motion information, if images are divided into a number of motion vectors, after computation, motion information are recorded for registration and interpolation.

Motion block and search range

The size of motion blocks is first set to 8 * 8 pixels, with a ± 16 -pixel distance search range, this is illustrated in Figure 11, which means that, each block will compare in the context up to two blocks shifting all directions, the comparison is to traversal in the search range, utilizing full search algorithm, after that, we could

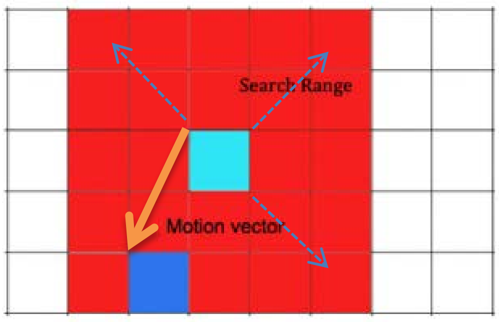


Figure 11 Sample Motion Block and search range demonstration

Block size: 8 * 8, search range: 40 * 40 get a reasonable motion vector indicates the smallest differences for this block. However, as experimented with HD images (1920 * 1080), such size configurations are relatively small, which shoots up computation, simultaneously, another element to consider in interpolation is accuracy, which could be affected by size as well, so I left the size changeable, and results are discussed in section 4.4.1.

With respect to the measurement of block differences, in each block comparing pair, I compute the MSD of total pixel pair differences, this algorithm is demonstrated below.

Algorithm 4 Block difference computation.

1) For one motion block in the input image, start with the first pixel in the block, assume the input image has m blocks, each block has n pixels.
2) Subtract the pixel grey value with corresponding pixel in the reference image, square the result D_p^1.
3) Add up D_p^1 to $D_p^n$of all pixels in the block as an illustration of block difference, denoting by D_b^1.
4) Traversal in the search range of this block from step 2) and 3), with step of 1 pixel, obtain D_b^1 to D_b^m.
5) Find the smallest $D_b^i,\ i = (1, 2, 3, \dots, m)$ from D_b^1 to D_b^m, record the motion vector (x_i, y_i) for this block.
6) Repeat from 1) to 5) until all blocks in the input images are computed.

Segmentation

At this stage, motion vectors have been computed and then they should be categorized into two parts, background and foreground. In time – lapse photography, the foreground parts usually contain moving objects, and for photographer, camera will be carefully installed to avoid shifting, so an assumption here is that the background will move slower (own smaller vector norms) than the foreground, so that there will be a threshold for the motions.

It is noticeable that with algorithm 4, consider the background part, even if two images are aligned perfectly, the slight intensity difference might not result in a motion vector of (0, 0), hence, a key element for detecting the threshold is that it should be somehow large to avoid such incorrect election probability, a mean value is less reliable since zero vectors could still account for a certain proportion. As a consequence, there vectors are neglected when selecting the threshold, below listed the algorithm.

```
Algorithm 5   Detection of motion vector threshold.

    1) Input all the motion vectors in algorithm 4, form a sequence
       $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$, where $m$ is the quantity of motion
       blocks.
    2) For $(x_i, y_i)$, $i = (1, 2, 3, \dots, m)$, compute $d_i = norm(x_i, y_i)$, $d_i$
       denotes the moving distance of a block.
    3) Exclude all vectors when $d_i = 0$, and sum up other non – zero
       distances, with whose quantity $N$ accumulated, results in $S$.
    4) Threshold $= S/N$ .
```

Now the threshold could be larger because zeros are intentionally disregarded, and then remaining data will result in a mean that could implement the segmentation, the vector norms of foreground blocks will be larger than T, while the remaining belong to the background, an example result of segmentation is shown in Figure 12.It could be seen that the

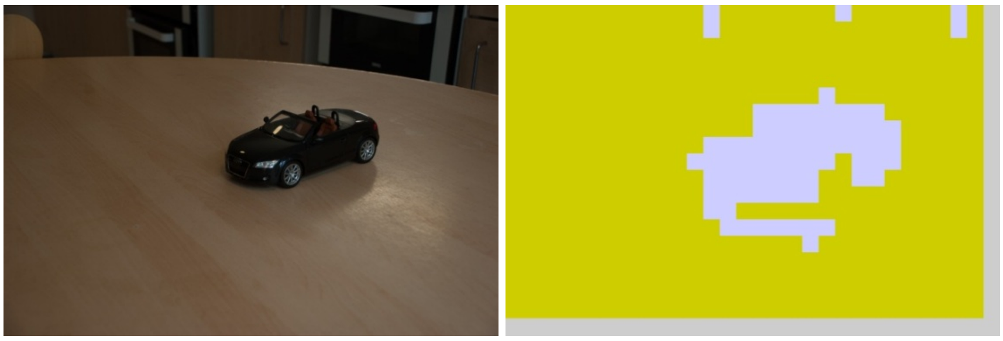


Figure 12 Sample segmentation by motion estimation

Left: Source image Right: Segmented result

segmentation could roughly recognize the background (yellow part), but via data selection before shift, such errors are under tolerance.

Shift the image

In the practice of time – lapse photography, the moving parts should not be considered when applying alignment operation, and this is the reason why I did the previous segmentation part, thus the shift vector (I_x, I_y) for the input image is actually the background 'motion' (compare the current frame to the reference frame) and (0, 0) indicates that registration is not required, I select the motion vector that owns the largest frequency in the background part, this not only neglect a few computation errors that could be caused by intensity changes, but enhance the probability that the background position relationship between the assessed image and the reference satisfies the selected vector. In addition, since image pyramid is employed for acceleration, the vector should be multiplied by 2^L if source images are shrunk L level(s) to a smaller size.

Acceleration

As demonstrated before, image pyramid is an effective method for acceleration, it equals to enlarge the block size in when using BMA algorithm, however, the level should be controlled to an extent for a relative reliable results, which is shown in section 4.4.1.

3.1.3 Summary

In this part I employed two methods for image registration, the first one is done by an XOR operator and is simpler, but after experiments, it performed less robust, the second has a segmentation of background and foreground, which is an inherent character for time – lapse photography, the results turn out to be pleased.

3.2 Interpolation

3.2.1 Motion Estimation for Interpolation

An exciting point from motion estimation method is the segmentation results, which could be extended to interpolation, but this should under the condition that images have been aligned, otherwise, it could gain computation for registration despite of a successful interpolation.

Motion vector selection

As input to the interpolation algorithm are aligned images, only the motion blocks from foreground (vector norms larger than threshold) are considered, similarly, I count frequencies for all vectors and the largest is regarded as the motion information of the object.

Creation of intermediate frames

Now that we have computed how much the objects moves from one frame to another, the intermediate position could be estimated, firstly, the motions are considered not large, so that the problem could be simplified as linear, if one image is need to be interpolated, the new position should at half way from the previous frame to the next one, then at these positions, pixels could be copied from either the previous or next frame.

Accounting for the background part, which is considered as static, I set the mean values of the neighboring two frames since they are calibrated and will not cause offsets.

In addition, there could be two methods for creating more images. First, repeat previous process, and the second could been implemented when new position calculation, if one more images is needed, for example, two positions should be computed by a step of one third from reference frame to current frame rather than half of that. However, this is out of project scope so I did not apply experiments.

Occlusion handling

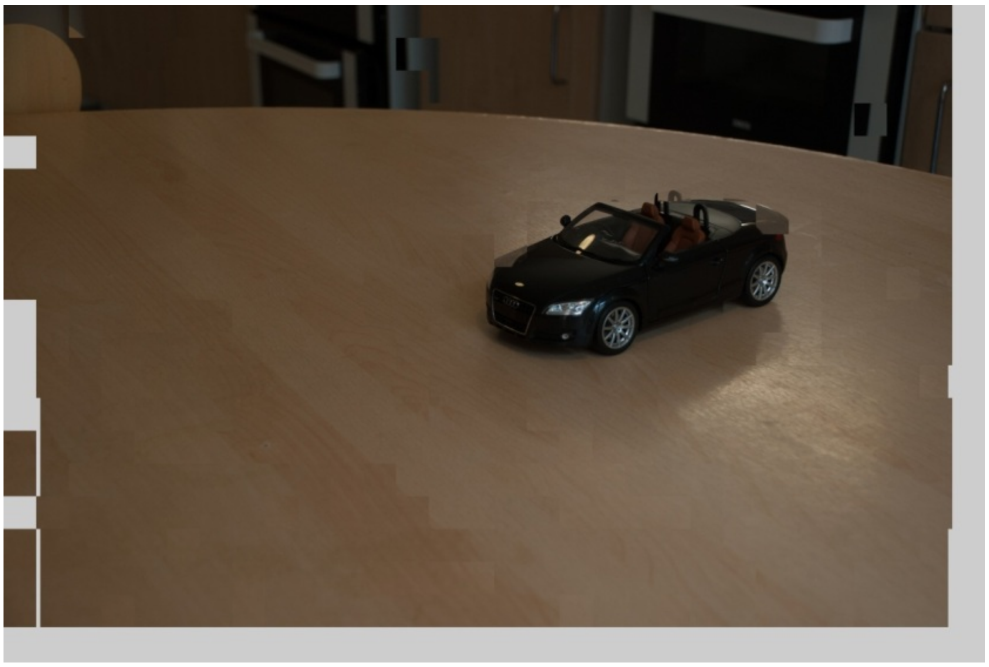


Figure 13 Empty pixels after interpolation

In image interpolation problems, there will be empty pixels in the newly created frames, which is displayed in Figure 13, as part of the edge is not being filled. Particularly, when previous method is applied, there could be holes at some edges because the input images are regarded as a 2 – D array, when processing the blocks near the edges, there will be out – of – bound problems, which have been purposely disregarded when comparing blocks, it is reliable because after that the motion vectors with largest frequency is forecast as the actual motion, therefore, empty pixels in the new frame is filled the same way as copying background

Algorithm overview

As described before, Algorithm 6 gives the process from selected motion vectors. It is currently designed for creating one image between images, if more novel frames are required, the process could be repeated.

Algorithm 6 Novel frame creation (Considered for one image).

1) Assume the foreground contains n motion blocks with n motion vectors, $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
2) Select the vectors that owns largest frequency and estimated as motion vector, demoted by (M_x, M_y).
3) New position coordinates $(x, y) = (x_1 - {}^{M_x}/_2,\ y_1 - {}^{M_y}/_2)$,

   where x_1, y_1 are the pixels positions in the current image, or $(x, y) = (x_0 + {}^{M_x}/_2,\ y_0 + {}^{M_y}/_2)$, where x_0, y_0 denote the

   pixels positions in the reference image.
4) Copy pixels from the corresponding coordinates in current or reference frame.
5) For values of pixel (x, y) in the background part of the new frame, $f(x, y) = {}^{f(x_0 + x_1, y_0 + y_1)}/_2$, where f denotes the pixel

   values of a pixel, note that the value here is in RGB colour channels.
6) Fill empty pixels with value calculation similar to 5).

3.2.2 Summary

Motion estimation offers a linear and illustrative method for image interpolation, and if the correct positions of moving objects are able to achieve, plausible new frame could be obtained by copying pixels from source frames, in addition, it is indispensable to fill blank pixels caused by this method.

3.3 Image Sequence Consideration

Previous methods still work for two images, it should be extend to the entire image sequence to create totally automatic program, only in this way, users are able to select image sequence and wait for their results. In this section, both registration and interpolation methods are developed for image sequence, but for the threshold and XOR algorithm, because it is difficult to determine whether two images are aligned and hence less possible to find a correct reference point, as a consequence, it has been categorized as an unsuccessful attempt in this project.

3.3.1 Find Reference Point for Registration

For two images to be aligned, either could be selected as the reference image, but in an image sequence, it is completely different, a correct reference point could massively save computation amount, for example, consider an image sequence consisting of 100 images that have been captured via a tripod, if there were slightly movement after capturing the 5th image, the fastest registration should apply calibration to the first five images rather than the remaining 95, hence any one from image 6 to 100 could be selected as a reference, if unfortunately image 1 to 5 is selected, the registration algorithm should be run for 95 times, thus it is significant to detect a best reference image, considering the speed issue in image sequence.

Assumptions

The assumption for reference point detection is formed from time – lapse practice, usually a photographer will be careful with the tripod to avoid movement, but it is almost impossible to re – install the tripod to exactly the same position after movement, this could be true when the process covers a long time, three days, for example, the photographer could mark the first position but it is still difficult to ensure perfectly aligned images, and that is the reason why I do registration. Under such consideration, it could be assumed that the input images are actually divided into n groups, images in each group are perfectly aligned because of usage of tripod (it could also be possible that only one image is in the group). To write the assumption into mathematical form, for all M images in image sequence S, they are divided into n groups, which could be denoted by a set $(G_1,\ G_2, \dots, G_n)$, their corresponding image quantities is written as $(M_1, M_2, \dots, M_n)$.

Main process

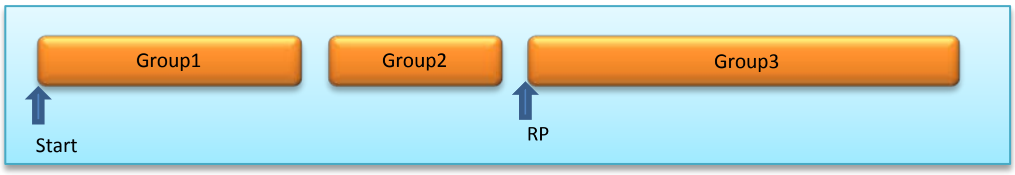


Figure 14 An illustration of reference point detection

Figure 14 gives an example of three image groups compromising an image sequence, the algorithm starts from the first image in the first group, selecting the first image as reference point, then time, check one image afterwards, if the image pair is aligned, an accumulator indicating the quantity of images that are aligned with the reference will increase by one, otherwise, record the quantity, move the reference point the first unaligned image, then repeat this process until all the images have been checked. At this stage, the images are grouped with each group has an aligned image numbers, the reference point will be the first image of the group containing the largest quantity.

Condition of alignment

At this point, I found the thresholding – XOR algorithm less reliable for time – lapse images because the motion could cause large errors, the smallest difference in entire image may not indicate the right offsets, thus this algorithm is abandoned and regarded as a failure in this project, in contrast, motion estimation could generate a correct result, when comparing only in the background part, if most blocks return vectors (0, 0), this background could be estimated as still, representing the alignment.

Algorithm description

The following box demonstrates the entire algorithm.

Algorithm 7 Reference point detection

1) For Groups $(G_1,\ G_2, \dots, G_n)$ in image sequence S in total N images, aligned image quantities for each group are written as $(M_1, M_2, \dots, M_n)$ (to be detected).Set the first image as the reference point, set up an image quantity accumulator acc.
2) Check one image each time after the reference point, if image $f_i, (i = 2, 3, \dots, N)$ satisfies the aligned condition, $acc =\ acc + 1$, then check the next image.
3) If two images are not aligned at image f_i, store acc to $M_j, (j = 1, 2, \dots, n)$, then move the reference point to $f_i,\ acc = 0$, repeat from 2) until the last one.
4) Find the largest M_j, set the final reference point to the first image of G_j.

3.3.2 Determine the Interpolation Points

Similarly to registration, not all the gap between two frames requires interpolation, only these are sufficiently large will apply the algorithm. From a motion vector (x, y), the distance of motion $\mathrm{d_m}$ could be calculated as $\mathrm{d_m} = \mathrm{sqrt}(\mathrm{x}^2 + \mathrm{y}^2)$. For an image sequence, I first detect all the distances between frames, then select the points that are within a specified range (set to 0.8 in test, could be set by user in practice) of the largest distance. Algorithm 8 in the next table describes this process.

Algorithm 8 Interpolation point detection

1) For image sequence S with images $(I_1, I_2, \dots, I_N)$, start from the first pair (I_1, I_2) (index = 1).
2) Compute the motion information and get the motion distance d_m^1, then index = index + 1, repeat 2) until the last pair.
3) Find the largest $\mathrm{d}_m^i, i = 1, 2, \dots, N$, set as . d_{max}.
4) Restart from the first pair, if $\mathrm{d}_m^i >\ 0.8\ *\ \mathrm{d}_{max}$, do interpolation between this pair.

3.3.3 Summary

When extend the methods to the entire image sequence, for registration, it is significant to consider the speed issue, a best reference must be selected for the sequence since it could save enormous amount of computation, with regard to image interpolation, the magnitude for creating new frame could be selected by user, smaller range will limit the condition of interpolation and thus accelerating.

3.4 Methods Summary

In section 3, all the methods and algorithms that are employed in this project have been demonstrated in detail with figures, firstly, registration and interpolation algorithms are applied for only two images for testing the effectiveness and robustness, among there is a failure which employs thresholding and then compares images with an XOR operator. Then, comparison is implemented before a background and foreground segmentation which is done by motion estimation, it is exciting when motion estimation is positive with both image operations. Finally, the methods are extended to image sequence and could be able to run automatically.

4. Implementation and results

The 4th section illustrates the results I have achieved with some implementation methods, in section 4.1 some data storing structures and objects are introduced, followed by some failure examples with analysis, then, motion estimation method results are shown in the next two sections respectively.

4.1 Introduction

The entire project is implemented by C++ using Open Source Computer Vision (OpenCV), program environment is a 32bit Windows 7 Running on a virtual machine, with Microsoft Visual Studio 2010. Min configuration of this virtual system is listed in the following table.

Table 1 Test machine configuration

Component	**Models**
CPU	Intel(R) Core(TM) i7 – 2620M @ 2.70GHz, 4 processors
Memory	3GB @ 1333MHz
Hard Disk	80GB
Graphic Card	Intel HD 3000

Cameras used

- Panasonic LUMIX DMC – LX5GK
- Nikon DSLR D80

The implementation of this project should first consider the image sequence problem, thus in this section, firstly vectors are introduced for storing image names, queue are utilized to store motion vectors, motion distances, which are described in section 4.1.2.

4.1.1 Image Names

Due to the input to my program is an image sequence, and OpenCV can easily read images by names, I create string - typed vectors to store image names, the only parameter that left to the user is the address of image sequence directory.

After the directory is specified, my program can search all files inside and record all their names (appears with directory) into a string vector instance called ‘srcFileNames’, simultaneously, a symbol string ‘aligned_’ is attached to each image name and they form another string vector ‘dstFileNames’, which is used for writing aligned output images. All input images have an output including their offsets is (0, 0), in such case, images are aligned to the reference so that they are only copied, which mainly boils down to the convenience for showing images in the program.

For showing the results of alignment, I write codes to overlay images with transparency reduced, in this way aligned images would be similar to the effect of showing a single image.

For image interpolation, there are slightly differences for output file names, when image pairs are computed for new frames, string prefix ‘inter_’ will be attached to the name of reference image indicating that it is a newly created frame, for example, file name ’inter_0001.jpg’ is the frame interpolated between images ‘0001.jpg’ and ‘0002.jpg’.

4.1.2 Data Structure

Queue are mostly used to store motion vectors, to accumulate distance frequencies for data optimization, as shown in Figure 15, three structures are designed.

- **struct blockMotionVector**

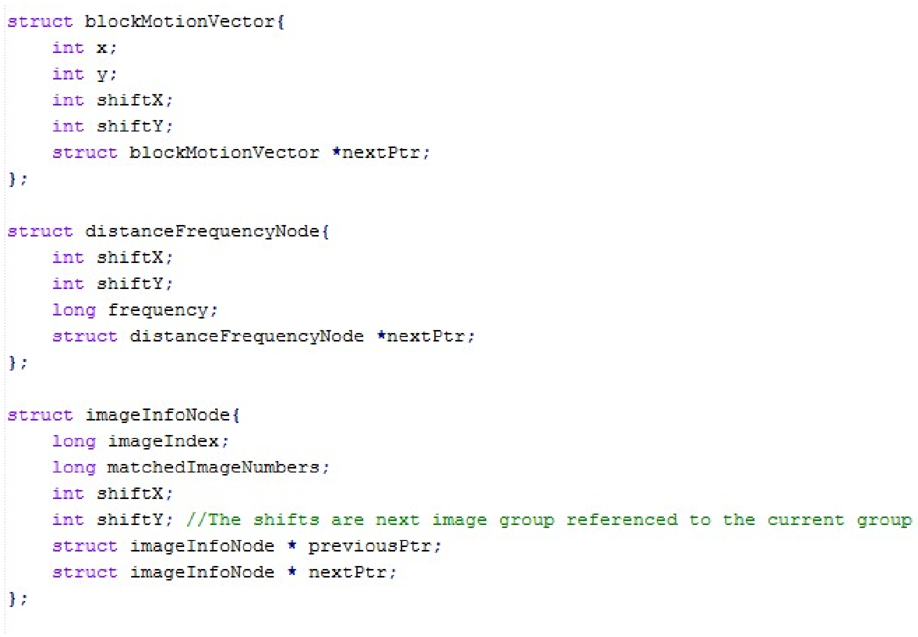
  For storing motion vectors when comparing two images, (x, y) is an index of motion blocks, $(shiftX, shiftY)$ denotes the motion vector.

- **strct disranceFrequencyNode**

  Used for counting frequencies of motion vectors.

- **struct imageInfoNode**

Only used for registration, this structure corresponds to the first images of each group, $imageIndex$ represents the image position with quantities of matched images stores in $matchedImageNumbers$, the nodes form a bi – direction queue since the reference point need to reference to both nodes in the contexts.



```
struct blockMotionVector{
    int x;
    int y;
    int shiftX;
    int shiftY;
    struct blockMotionVector *nextPtr;
};

struct distanceFrequencyNode{
    int shiftX;
    int shiftY;
    long frequency;
    struct distanceFrequencyNode *nextPtr;
};

struct imageInfoNode{
    long imageIndex;
    long matchedImageNumbers;
    int shiftX;
    int shiftY; //The shifts are next image group referenced to the current group
    struct imageInfoNode * previousPtr;
    struct imageInfoNode * nextPtr;
};
```

Figure 15 Structures definition

Due to the First – In – First – Out (FIFO) character of queue, the motion information could be accessed strictly as the sequence has been stored, this is extremely significant for registration in image sequence, when file names are in sequence stored in vectors and could be accessed by an index, in that way, the elements in the vectors could be projected to nodes from $imageInfoNode$, so that each image could easily get an shift vector when applying shifting operation.

4.2 Unsuccessful Results

In this section, some results that are not pleased but significant to the project are illustrated, including failure examples of thresholding – XOR algorithm and some error in image interpolation.

4.2.1 XOR Registration

Registration by comparing the differences of entire images could obtain positive results, but less reliable when motion is presented in images, a expressive example of this algorithm is shown in Figure 16, images are captured by the LUMIX in Edinburgh, with hand – held exposures, it can be seen that buildings are decently aligned in spite of the clouds in the sky, in the right side of the image, the bell tower is failed because of the massive distortion of the lens, which is set on a focal length of 24mm equivalent to a full – frame system.

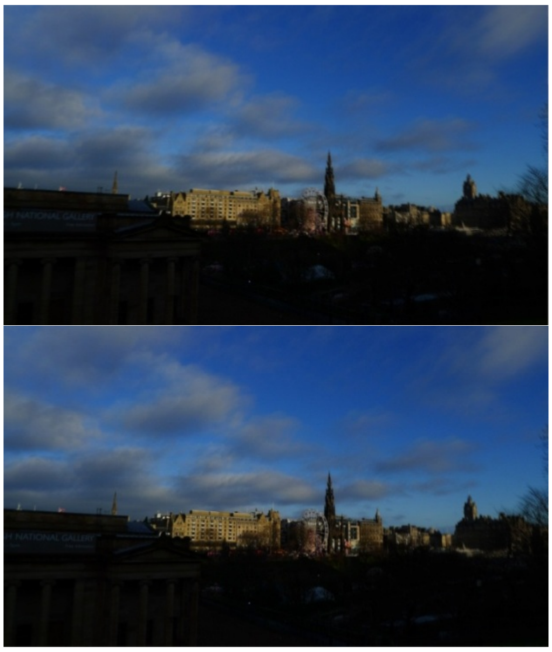


Figure 16 A result aligned by the thresholding – XOR algorithm

Figure 17 Failure example of the XOR computation

In contrast, the failure in Figure 17 shows almost no differences before and after registration, the model car is actually moving and should be neglected, but the background still kept displacements, I experimented with several other examples but unfortunately obtain similar results. It can be seen that the XOR algorithm is less robust, in further work, it could still be difficult to find a reasonable standard to determine whether two images are aligned and then apply shifting to images, as a consequence, this part of work has been classified as unsuccessful category.

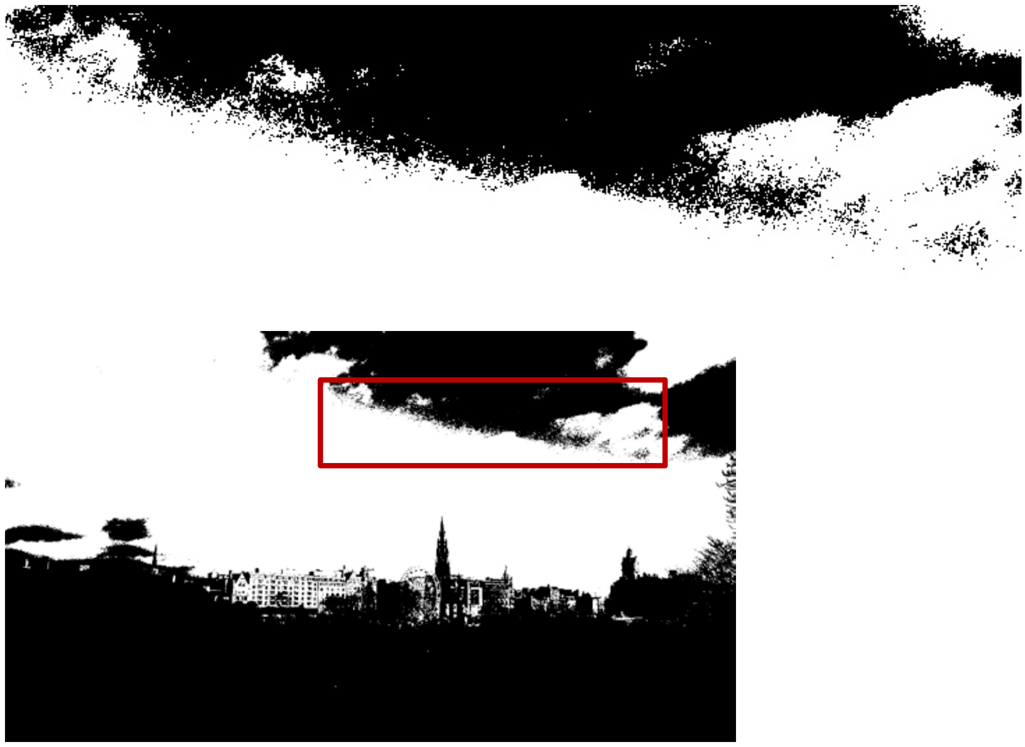


Figure 18 Noises near threshold

Figure 19 Exclusion image example

In addition, (G. Ward, 2003) proposed a method for handling noises with an exclusion image, which is comprised by 0's and 255's, pixels are set to 0 if grey values are within a certain distance of the threshold, otherwise, they are set to 255, in this way, the noises near threshold could be shown in white points in the exclusion image, Figure 18 illustrates the noises in the red square in the actual pixels, and Figure 19 is an example of exclusion image of the 2 – value image in Figure 18, this exclusion image is computed by an AND operator with the result from XOR calculation, but the result turned out to be no differences.

4.2.2 Summary

In conclusion, although the XOR algorithm could achieve some effect, when comparing with motion estimation based method, computation in the entire image shows obvious disadvantages, and considering the difficulty for sequence extension, this algorithm is waived in the later part of the project.

4.3 Motion Estimation Registration

In this section a number of images that are aligned by motion estimation algorithm, including experiments with only two images and an entire image sequence.

4.3.1 Results

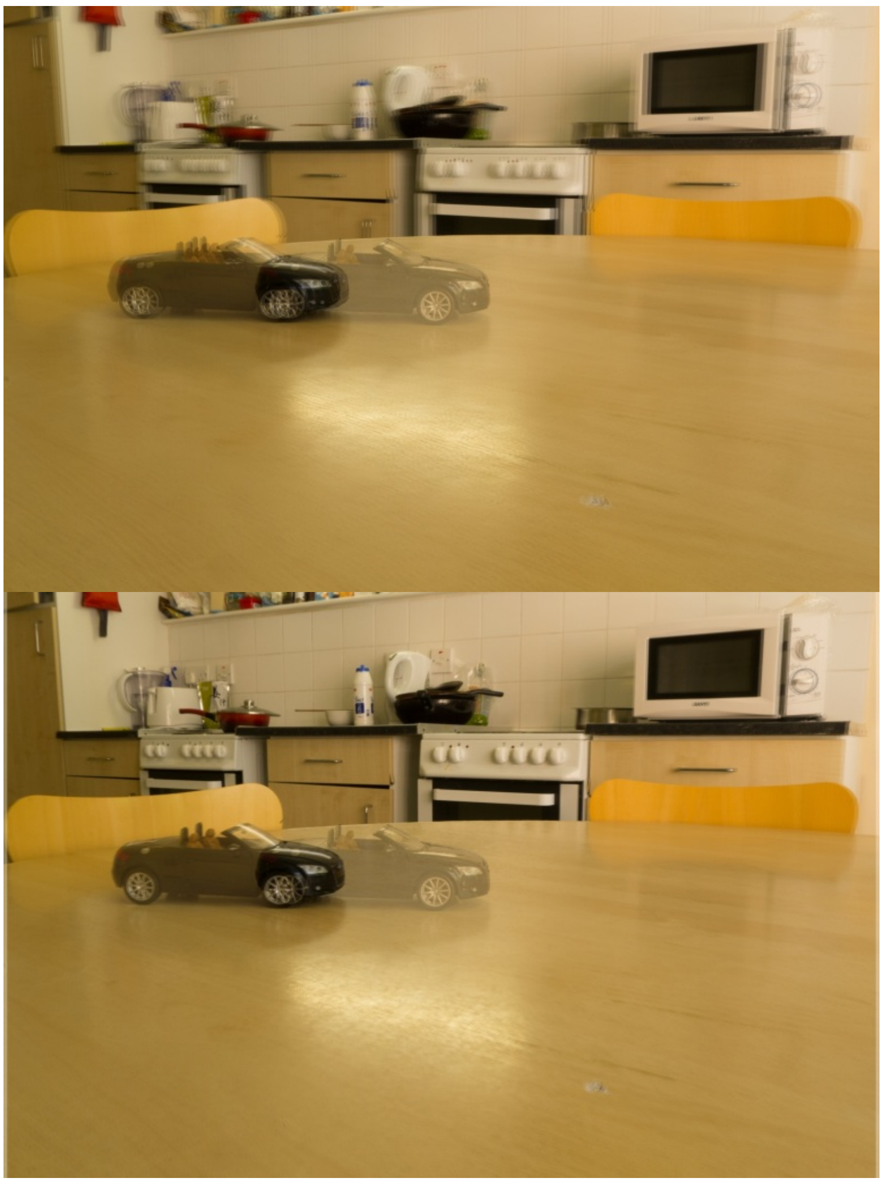


Figure 20 Success alignment for the failure example of the XOR computation

Up: Unaligned results Down: Aligned images

In figure 20, the result that is not achieved by the thresholding - XOR method is shown, it is noticeable that the background are reasonably aligned, but the source images still representing the motion, and hence the overlapped image has ghosts of the model car. Another representative example is demonstrated in Figure 21, where the trees and mountains have been obviously calibrated, comparing to the model car image, segmentation of images of Figure 21 could gain difficulty because the moving parts are actually the cloud, while background, where should be apply alignment are the mountains and trees, which requires the motion blocks to be smaller (large blocks could fail, because background takes smaller proportion of the image size), and Figure 21 is in similar condition to the previous Edinburgh images.

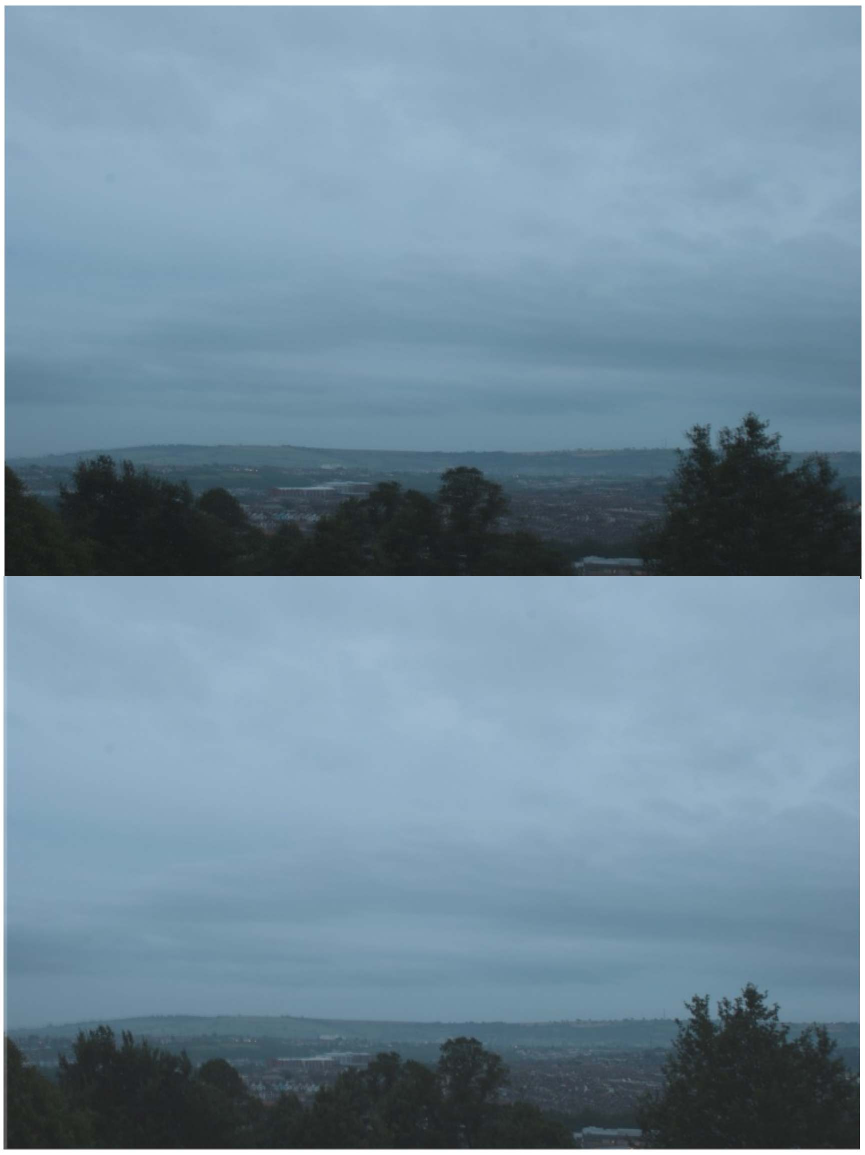
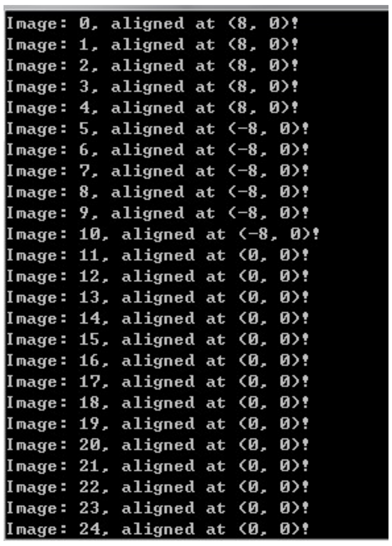


Figure 21 Another example aligned by motion estimation

Up: Unaligned results Down: Aligned images



```
Image: 0, aligned at (8, 0)!
Image: 1, aligned at (8, 0)!
Image: 2, aligned at (8, 0)!
Image: 3, aligned at (8, 0)!
Image: 4, aligned at (8, 0)!
Image: 5, aligned at (-8, 0)!
Image: 6, aligned at (-8, 0)!
Image: 7, aligned at (-8, 0)!
Image: 8, aligned at (-8, 0)!
Image: 9, aligned at (-8, 0)!
Image: 10, aligned at (-8, 0)!
Image: 11, aligned at (0, 0)!
Image: 12, aligned at (0, 0)!
Image: 13, aligned at (0, 0)!
Image: 14, aligned at (0, 0)!
Image: 15, aligned at (0, 0)!
Image: 16, aligned at (0, 0)!
Image: 17, aligned at (0, 0)!
Image: 18, aligned at (0, 0)!
Image: 19, aligned at (0, 0)!
Image: 20, aligned at (0, 0)!
Image: 21, aligned at (0, 0)!
Image: 22, aligned at (0, 0)!
Image: 23, aligned at (0, 0)!
Image: 24, aligned at (0, 0)!
```

Figure 22 Image sequence registration information output

The Figures have expressed the effectiveness of motion estimation method, when applied to image sequence, it could still detect the correct image reference point and apply registration, Figure 22 illustrates a result when experiment with an image sequence, which is purposely formed as Figure 14, Group 1, 2, 3 contains 5, 6, and 14 images respectively, and image index, relatively, starts from 0 to 24, the camera displacement is intentionally set and the program achieve an expected result with the reference image selected as image 11, thus only 11 images (image 0 to 10) requires registration rather than 14 if simply set the first image as reference. As experimented, each registration operation could cost up to roughly 1 second, therefore, it could be forecast that the computation would be massively reduced when the image sequence contains larger number of images.

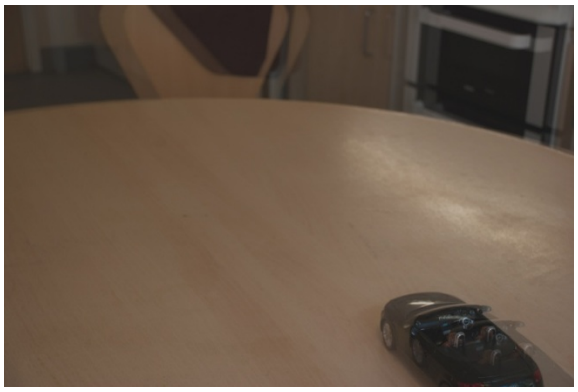


Figure 23 Failure of motion estimation

An unsuccessful example is shown in Figure 23, in there two frames, camera movement becomes abnormally large, which is just very close to the car model, but they move towards different directions, so that the segmentation is failed and thus has negative effects on image shifting.

4.3.2 Summary

The motion estimation method, due to a background and foreground segmentation, could perform better than single XOR operation in the image, it could work on images with apparent background and moving objects, as well as wide angle scenes, but only for some efficiency differences, one situation that fails the algorithm is that the background and objects moves in different directions but in very similar distance.

4.4 Interpolation

The interpolation results are shown in this section with comparisons between different parameter configurations, the unpleased results in the early stages are also expressed as a significant process to obtain plausible new frames.

4.4.1 Results

Motion block selection

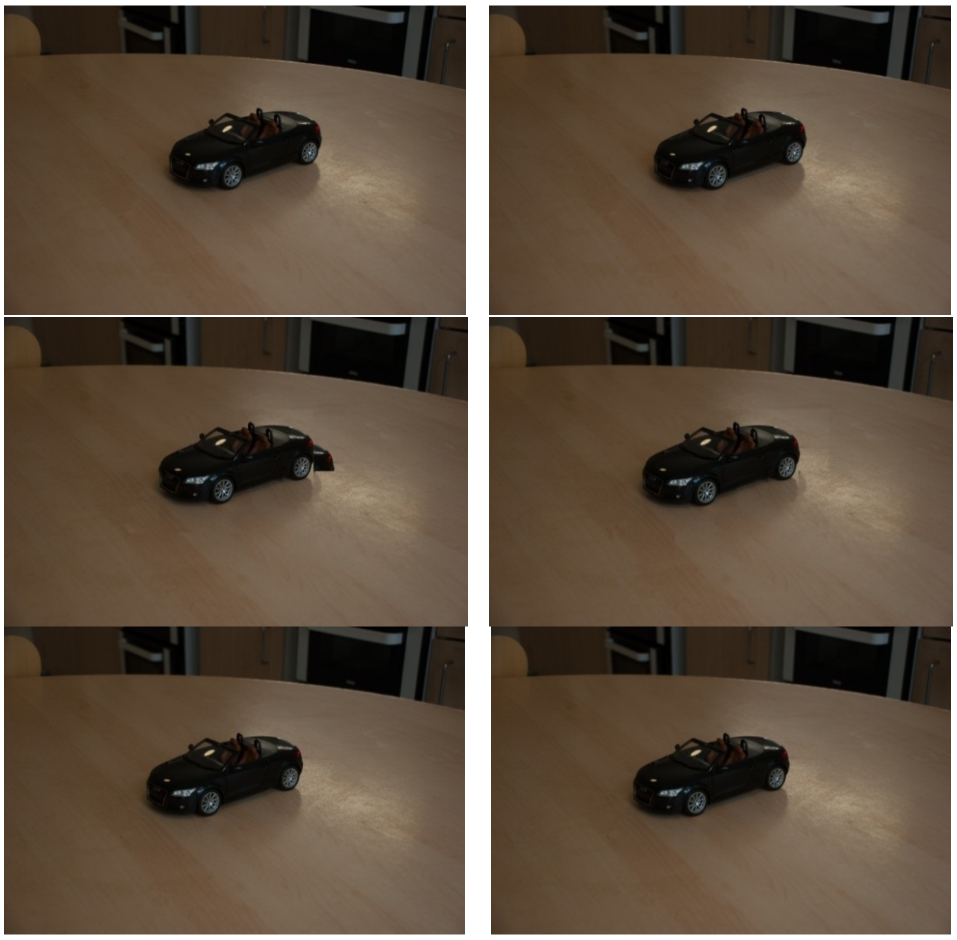


Figure 24 Comparison before and after motion vector selection

Left: No selections Right: Selection by frequency

In the early stage, after motion vectors are recorded, the new frame is computed by the raw data, which could result in incorrect position of some blocks, as shown in Figure 24 (left), a block containing the trunk of the car model located in the background and cause an obvious error, but the main body of the car is just in the correct position, it can be seen that the most motion blocks obtain the reasonable motion information apart from a few exceptions, which result in errors, from this example I tried to select the motion vectors from frequency, use the motion vectors with largest frequency to represent the entire object, by implementing that, incorrect motion vectors are re – assigned with values from most motion vectors, thus avoid improper motion blocks. A comparison of images after motion vector selection is displayed in the right part of Figure 24.

Motion block size

It is difficult to specify a motion block size, experiments show that large motion blocks could reduce the computation amount and hence increase speed, but some part of the objects could be under – segmentation, to put another way, the foreground could be unexpectedly classified as background. However, smaller blocks make contributions to accuracy, since images are sufficiently divided and motion blocks are moving to the correct position. Figure 25 demonstrates an interpolation at the same point with different block sizes, in this image, block size are actually not changed but the image pyramid is modified as an alternative. Both operations are equivalent because the computation varies linearly. In the upper image, the size is larger, so one block containing the part of left back wheel is classified as background and hence kept still. In contrast, when reduce one level in pyramid, the motion blocks equally shrunk to a quarter of previous area, the car is sufficiently divided as foreground. Furthermore, if the pyramid level continues goes up, smaller

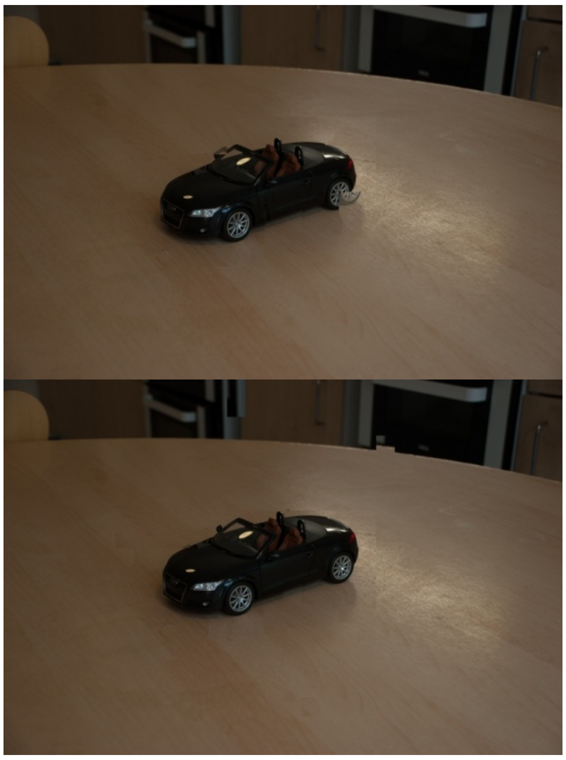


Figure 25 Different block size comparison

Block size: 8 * 8

Search range: 40 * 40

Pyramid level of top image: 4

Pyramid level of bottom image: 3

Pyramid change equals the effects of resizing the block and search range because images are linearly shrunk to a quarter size of one lever higher motion blocks could result in extremely negative effect, firstly the computation workload could shoot up due to such square – magnitude growth and secondly, over – segmentation would be presented, since the difference of block is calculated from grey values, slightly difference of intensity could cause the problem. Figure 26 is an unsuccessful interpolation result obtained in a long time beyond tolerance. Hence, a reasonable pyramid configuration becomes significant. With a number of experiments, I found that a reasonable configuration could be 8 * 8 block size with 40 * 40 search range, and a pyramid level of 3 and 4, under such conditions, a relative higher probability of obtaining plausible images could be achieved. Comparisons are shown in Figure 27 and Figure 28, same sequence are input to the program, when pyramid level is set to 4, four images are created and three are plausible, when the pyramid level reduced by 1, five

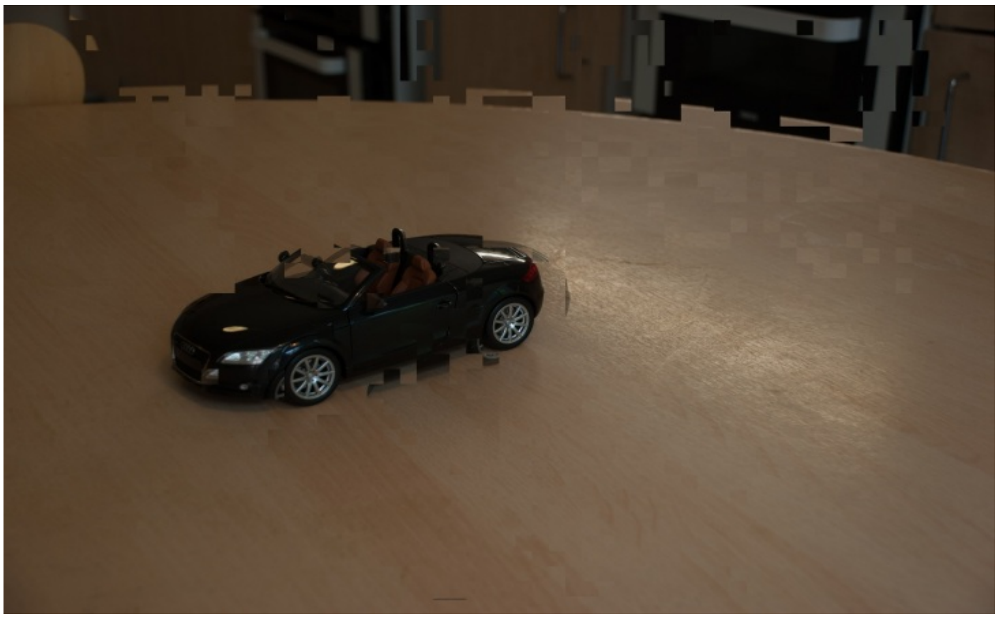


Figure 26 Different block size comparison

images are created and the result is slightly improved.

Interpolation point

For an image sequence, the interpolation is applied wherever the motion distance between two frames are within a specified range of the largest one, the range is set to 0.8 in experiments, and various number of images will be created, Figure 28 is an example that five images are created, compared to the results shown in Figure 27, the only difference at the beginning of the program is the level of image pyramid.

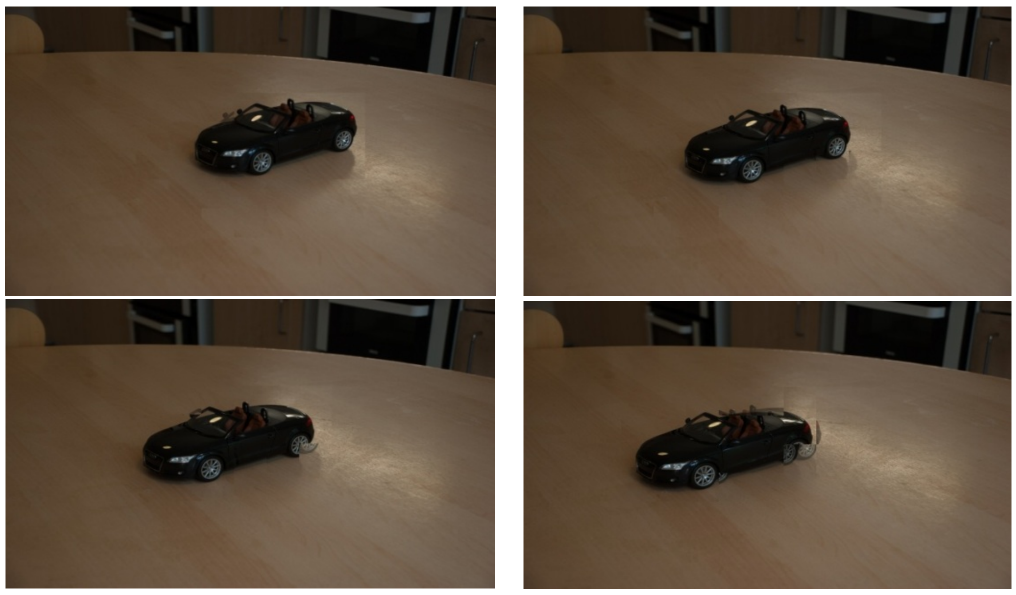


Figure 27 Results from image sequence interpolation

Pyramid level: 4

Except for the last frame, other images could be regarded as plausible, the model car is in the correct position while segmentation is reasonable, interpolation for the 4th image is a bit more difficult, motion blocks are relative large for this context, it is improved in the case of Figure 28.

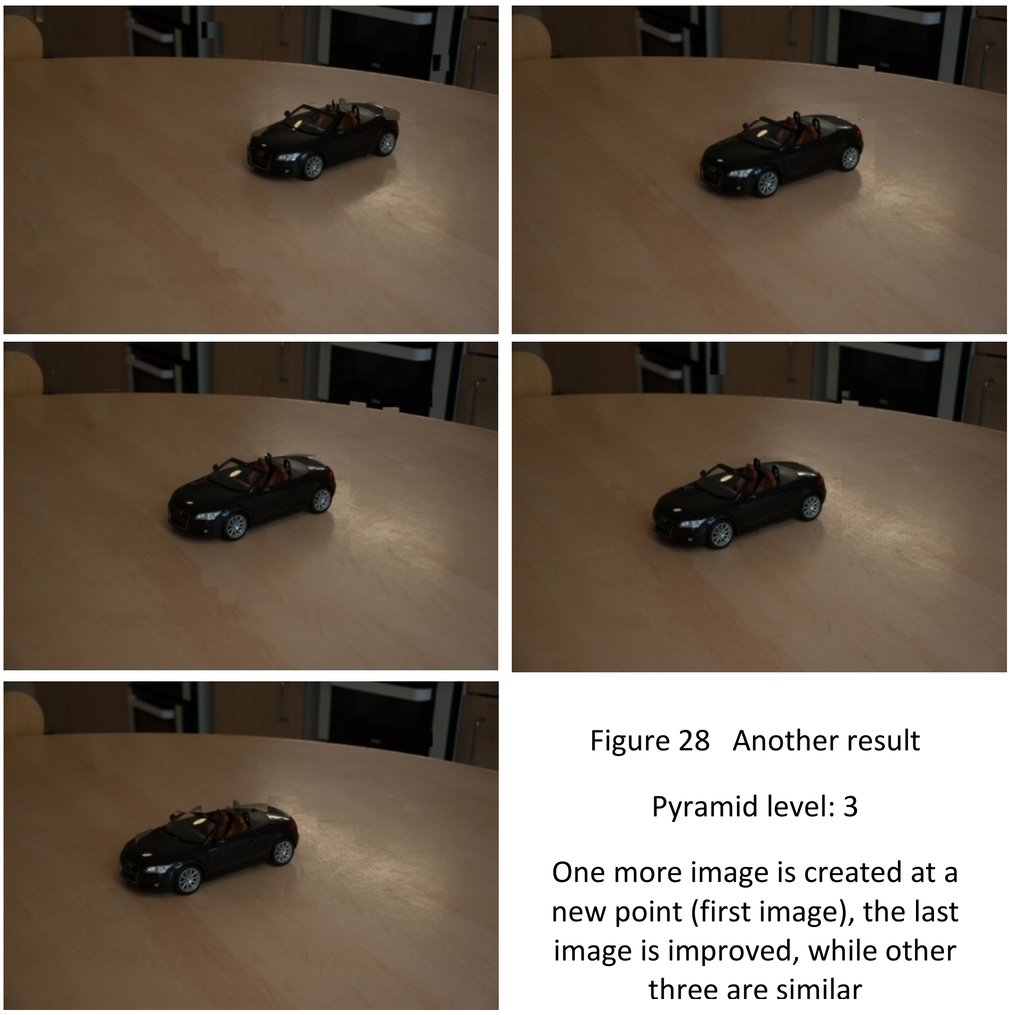


Figure 28 Another result

Pyramid level: 3

One more image is created at a new point (first image), the last image is improved, while other three are similar

4.4.2 Summary

In this program for interpolation, after images sequence is input, process is completed automatically and acceptable results have been achieved, there are several significant elements that could affect the result, first, optimization of motion vectors could avoid individual incorrect motion block movements Second, adjusting motion block size is a trade – off between speed and accuracy, large block sizes gives fast algorithm with less accuracy, the result could be under segmented, while vice versa. Finally, the condition of interpolation could be left to the user, using the largest motion as a measurement, could create images between large motions, which contributes to a detailed story telling when images are projected to films due to an equivalent effect of reducing playing back rate.

Some more images that have been tested are shown in the section 4.5.

4.5 Additional Tested Results

4.5.1 Registration Results

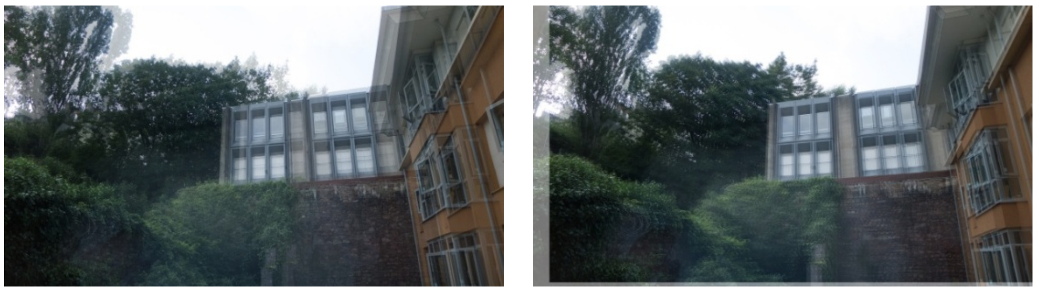


Figure 29 A not perfect result

Figure 29 shows a close but not perfect registration result, it can be seen that the windows in the center part and buildings in the right of the images are nearly but not strictly aligned, actually this is due to a severe camera rotation.

Figure 30, comparing to the previous example, camera are controlled only in translation, when capturing images, hence, the result is much better, the blur in Figure 30 may due to long exposure.

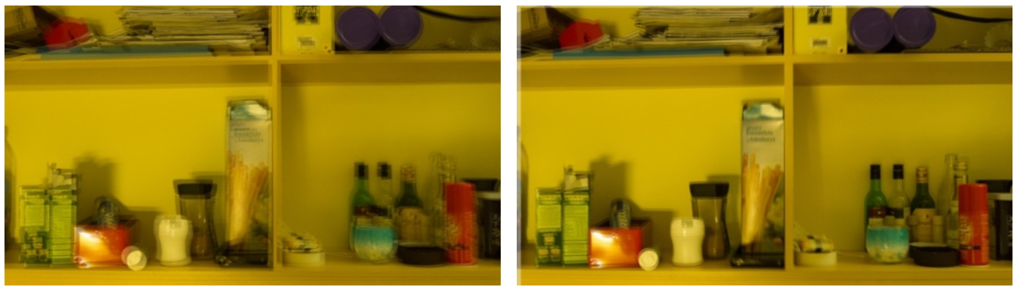


Figure 30 Better result when camera are not rotated

Examples in Figure 29 and 30 illustrate that the algorithm could not only work for images presented with motions, but suitable for relatively 'static' images, actually the entire image are regarded as background but it motion vectors are selected in the same way. As a consequence, it demonstrates from another aspect that motion estimation based method could neglect the motions in source images, which is ideal for time – lapse photography.

4.5.2 Interpolation Results

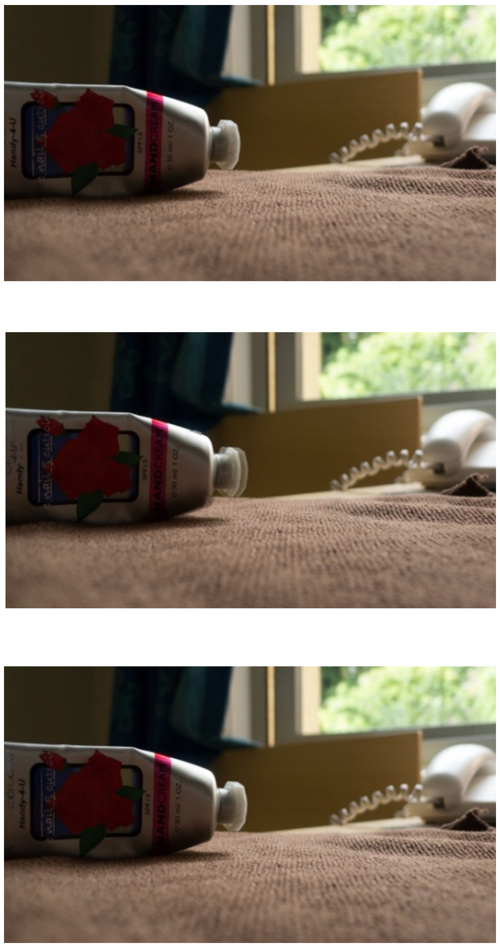


Figure 31 A result that is close to plausible

(Middle image are newly computed)

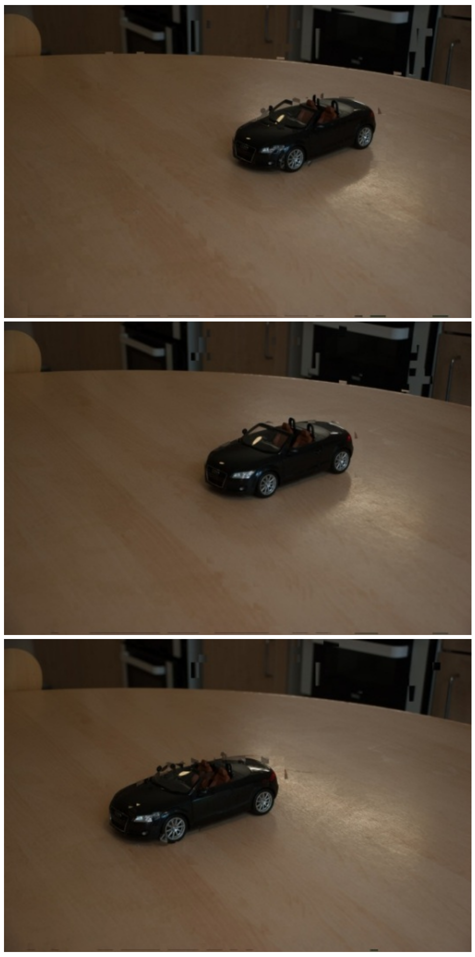


Figure 32 Unpleased results of excessive small block size

Small block size could give more accuracy, however images could be over segmented, some blocks in the background are categorized as foreground.

The results given in Figure 31 and Figure 32 not perfectly interpolated, in Figure 31, the cream cap is difficult to be contained in a block that is categorized to the foreground, thus it is incorrectly segmented to the background, in addition, in the very left part in the image, pixels are almost the same so that the letters are not decently recognized.

The three images in Figure 32 are all interpolated from source images, it is noticeable that there are error motions in the background, this is because of small motion blocks, which could result in over segmentation, so that some background elements are regarded as moving.

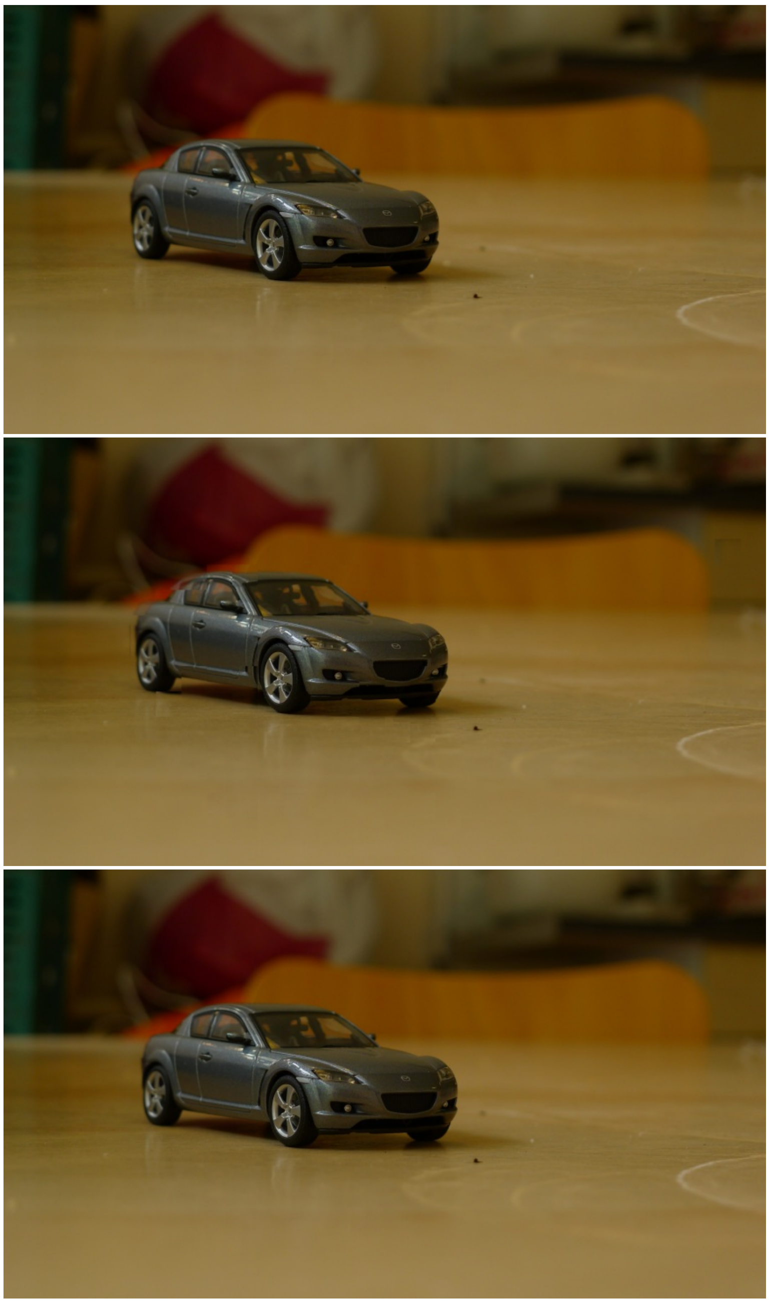


Figure 33 Another visual plausible result

Middle image are newly created

Figure 33 is a correct interpolation result with plausible visual effect, the motion of the car model is not significant, while the pixels in the intermediate frame could be projected to the right position.

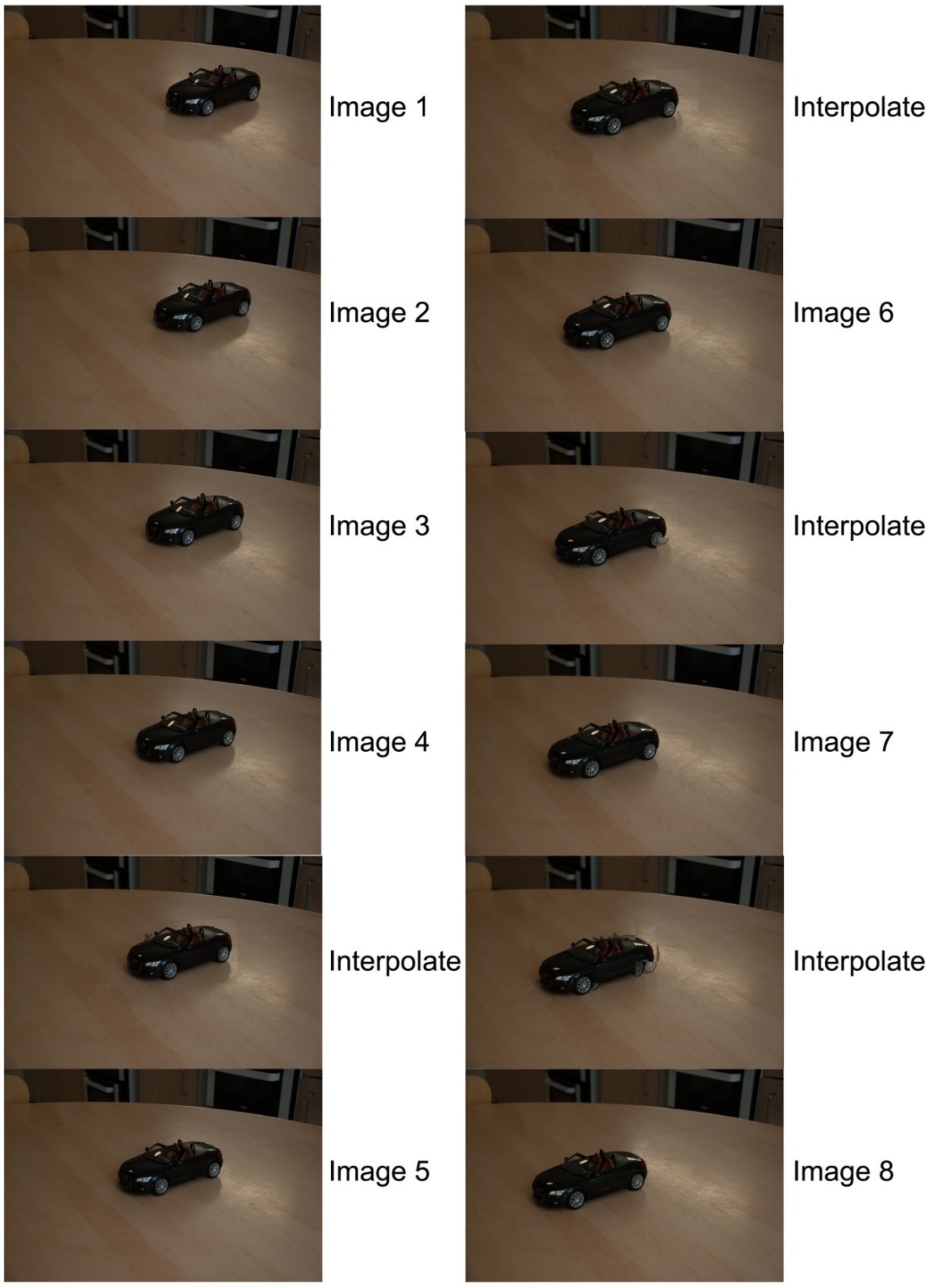


Figure 34 Result of image sequence interpolation

Figure 34 shows an example that interpolation is applied to an entire image sequence, the program can automatically search in a specified directory and then read each image pair one time, it can be seen from the graph that interpolated image could make compensation to relative large motions that are presented between frames, and the distance of car model movements become similar between frames after interpolation.

4.6 Results Summary

The failure of thresholding – XOR algorithm in this project leads to a decision making for smaller region segmentation, it can be seen that motion estimation could work for both registration and interpolation, a reasonable segmentation is the key factor to get successful results. In this project, the method achieves some decent images and is successfully applied to image sequence, hence, for time – lapse photographer, the operation for image sequence is completely automatic. However, algorithms still find some unsolved problems, for example, when background and foreground move to similar distance in two directions, it is fairly difficult for segmentation, and difficulty for interpolation is to find a proper block size.

5. Discussion

In this section the results observed are being discussed, with comparison when relevant, it will also give a conclusion to each method whether it is effective for this project. Finally, aims and objectives of the project will be reviewed and compared to what actually have been down.

5.1 The Failed Method

5.1.1 Reason Analysis

There are two methods employed for image registration part, the algorithm from (G. Ward, 2003) was firstly preferred boils down to previous HDR image processing experience, this method applies segmentation by a median threshold, which is the grey value from the intermediate position of the pixel after sorting by grey values, this could eliminate the effect from various intensities in the image. In HDR image construction, the intensity is completely different for three source images (usually use three images to represent details from specular highlight, darkness and intermediate grey parts in the scene), while for the sequence of time – lapse, the intensity is unpredictable, it could change massively when recording a sunrise process, for example, and sometimes it only fluctuates slightly. Hence, theoretically, after such segmentation, the algorithm could be applied for aligning images, however, the result turned out to be frustrated despite of some successful examples, the reason for that is that motions are usually presented in both input images (no motions in HDR source images), and to obtain offsets in x and y directions, two images are computed by an XOR operator, which is reluctant to ensure that the shifting vector is returned from a result containing least non – zero values, and furthermore, when applied into the image sequence, it is difficult to judge whether an image pair is aligned or not.

5.1.2 Further Thinking

After recognizing the causes of failure, it is necessary to find ways to avoid the influence from motion, since to compare differences on the entire image is not reasonable, in time – lapse practices, motions should actually be neglected when applying registration, ideally operations should only work with the background parts. Therefore, it is natural to divide images to small parts and separate motions from background, which is the reason why motion estimation method is finally implemented.

5.2 Motion Estimation

5.2.1 Comparison with Thresholding – XOR method

Motion estimation provides an algorithm to recognize the background and foreground, which inherently circumvent the problem from the thresholding – XOR method, background is recognized by motion vectors from motion blocks. The motion information of the background could be used for registration, while that of foreground indicates the object movements.

Each motion Block owns a motion vector predicting the position relationship between two images, among them some vectors are incorrectly estimated since the slightly changes in intensity, these could be neglect because most vectors are in the same motion, which is detected from the motion vectors that are in largest frequency, then the motion vector could represent the position relationship which is returned for shifting images. This is reliable because the background parts in the image are actually moving together when camera is slightly moved, the image is subdivided to motion blocks, comparing with the thresholding – XOR method, it has two main advantages. First, difference computations are operated only in backgrounds, second, since subdivisions are actually input into comparison, the genuine motion of the background is indicated by most motion blocks, as a consequence, the vector with largest proportion is selected, by doing that, larger sample volume is examined so that the probability of correct decision making is enlarged.

5.2.2 Motion Block Size and Image Pyramid Levels

Motion block size and image pyramid levels are two equivalent factors that could control speed and accuracy in the motion estimation method, excessively small motion blocks (or very low pyramid level) will lose both because of a substantial computation quantity in conjunction with over segmentation, on the opposite, very large motion blocks (or high pyramid level) could result in fast speed but under segmentation, thus it is significant to determine a equilibrium point, from the experiments carried out with HD (1920 * 1080) level, an acceptable point could be 8 * 8 block size with 40 * 40 (± 16) search range, image pyramid level is set at 3 (actual input image resolution 240 * 135).

5.2.3 Applicability

Motion estimation based method could be effective if an image is successfully segmented, throughout the experiments, for image registration, 50 image pairs are tested and 33 results could be accepted, including those are closely to perfect, while for interpolation, there are 30 reasonable results out of 50 input pairs, to sum up, a conclusion is shown in the following table.

Table 2 Experiment Results Summary

There are limitations for this method. Firstly, in our experiment, images with similar motions in background and foreground are difficult to be segmented and therefore less likely to obtain plausible registration and interpolation.

Registration			**Interpolation**		
Input pairs	**Perfect results**	**Reasonable results**	**Input pairs**	**Perfect results**	**Reasonable results**
50	24	9	50	15	16
Successful Proportion	0.66		**Successful Proportion**	0.62	

Secondly, since the method is designed for 2 – D motion field, problems in 3 – D motion field are not included.

5.3 Applied to Image Sequence

The most interesting and exciting point of this project is when the methods are extended to the entire image sequence, for image registration, a key point is to determine a reference image, because it could optimize the processing speed when the reference is the one to which majority images in the sequence are matched, otherwise, if the first one is selected or applying arbitrary selection, there might be massive computation cost. In addition, by implementing the reference detection, the entire image sequence could be automatically registered, which could contribute to time – lapse photographer since it is common to deal with thousands of images.

For the interpolation part, as interpolation is applied whenever a relative large motion is detected, more details are displayed in the film and hence the time become non – linear, it becomes more expressive when the photographer misses some motions when capturing images, similar to registration, in this project the interpolation is also automatic for the purposes of a more illustrative story expression.

In conclusion, an automatic registration method for image sequence has been implemented in this project and hence, it is convenient to process source images from time – lapse photography, it saves the time to find the images that requires alignment. Furthermore, time could be automatically adjusted when large motions are presented between two frames, this could offer more details to the audience.

5.4 Evaluation

Objectives review

- Objectives for image registration
    - Implementation of an algorithm to align two images.
    - Apply the algorithm to image sequence.
    - Experiment with the algorithm for acceleration.
- Objectives for image interpolation
    - Implement an algorithm which is able to create a new image between two neighboring frames.
    - Apply the algorithm to image sequence.
    - Experiment with the algorithm to enhance the plausibility of the new frame.
    - Find methods to automatically decide where to create new frames.

The first objective of this to implement registration between two images, and two algorithms have been experimented with, although the thresholding – XOR method is less robust, it made contributions to the beginning of exploring with the motion estimation method, which could extract the background (segmentation) and shift images as per the motion vectors based on image subdivisions, in addition, motion estimation could be applied to aligned images and compute with foreground to predict the motion of moving objects between frames, with occlusion handling, plausible new frames could be computed and this obtains the interpolation objective of this project.

Another significant objective is to extend the methods to image sequence and automatically implement registration and interpolation. Firstly, vector container in C++ is employed for recording all file names in a specified directory, in conjunction with creating new file names. Then vector objects are working with queue structure to detect reference image for registration, and to determine interpolation point between two frames. In short, this project has developed image processing techniques to fit a large number of images without any manual operation.

For acceleration, image pyramid is an effective and efficient to shrink images to smaller size with a quarter computation amount lessened when reducing a level, specifically, in my program it works with motion block size and search range which are together pre – defined, usually it could work at the found equilibrium combination and for software development consideration it could be selected by the user in case of some exceptions.

With regard to robustness, it is pleased to achieve a successful probability at 66 percent and 62 percent in registration and interpolation, when testing with 50 example image pairs respectively.

In conclusion, the key contribution in this project is the implementation of an automatic processing method for image sequence, which offers convenience in time – lapse photography.

5.5 Further Work

5.5.1 Extension of Interpolation

Comparing to image registration, interpolating operation requires more accuracy on motion estimation, it could fail when the motion object is not exactly segmented, at current stage, the interpolation could only be applied to 2 – D motions, and thus it is difficult to interpolate between frames where objects are scaled, or there are depth variations. As a consequence, one aspect in the future work could focus on interpolation at 3 – D motions, and so that suitable for more complicated situations.

5.5.2 Software considerations

As algorithms and methods in this project mainly benefits time – lapse photographers, who should interact with software interface rather than the programs, thus, from this aspect, relevant software could be developed based on these algorithms. Software could be either developed on PC machines or based on web browsers, more factors such as user – defined parameters should be taken into consideration.

6. References

[1] Iain E.Richardson,The H.264 Advanced Video Compression Standard John Wiley & Sons ltd, 978-0-470-51692-8, 2010.

[2] I.E.G. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia, John Wiley & Sons, Ltd, 0-470-84837-5, 2003.

[3] Alois M.Bock, Video Compression Systems from first Principles to Concatenated Codecs,The Institution of Engineering and Technology, 978-0-86431-963-6, 2009.

[4] Yu-Wen Huang, Ching-Yeh Chen, Chen-Han Tsai, Chun-Fu Shen And Liang-Gee Chen, Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results, Journal of VLSI Signal Processing , Springer Vol.42, pp.297–320, 2006.

[5] Alan Conrad Bovik, The Essential Guide to Video processing, Elsevier, 978-0-12-374456-2, 2009.

[6] Murali, E. Krishnan, E. Gangadharan and Nirmal P. Kumar, H.264 Motion Estimation and Applications, Video Compression, Dr. Amal Punchihewa (Ed.), ISBN: 978-953-51-0422-3, InTech, 2012.

[7] Aseem A, Maneesh A, Michael C, et al. Photographing Long Scenes with Multi-Viewpoint Panoramas. ACM Transactions on Graphics, 25(3), pp. 853-861, 2006.

[8] Loewke K, Camarillo D, Piyawattanametha W, et al. Realtime Image Mosaicing with a Hand-held Dual-axes Confocal Microscope. Progress in Biomedical Optics and Imaging. Proc of SPIE, Endoscopic microscopy III, Proceeding of SPIE, 6851: 68510F1-9, 2008.

[9] Heinze N, Esswein M, Krüger W, et al. Automatic Image Exploitation System for Small UAVs. Proc of SPIE on Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications, 6946: 69460G1-10, 2008.

[10] Q. Zhang, Z. Zhang, D. Zeng, A fast, Automatic and Robust Image Registration Algorithm, International Conference on Virtual Reality and Visualization, 2011.

[11] F. L. Seixas, L. S. Ochi, A. Conci, D. C. M. Saade, Image Registration Using Genetic Algorithms, Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008.

[12] G. Ward. Fast, Robust Image Registration for Compositing High Dynamic Range Photographs From Hand-held Exposures. Journal of Graphics Tools: JGT, 2003.

[13] D. Mahajan, F. Huang, W. Matusik, R. Ramamoorthi, P. Belhumeur Moving gradients: a path-based method for plausible image interpolation, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2009 TOG Homepage, Volume 28 Issue 3, 2009.

[14] T. Stich, C. Linz, C. Wallraven, D. Cunningham, M. Magnor, Perception-motivated Interpolation of Image Sequences, ACM Transactions on Applied Perception: TAP, 2011.

[15] C. L. ZITNICK, N. JOJIC, S. B. KANG, Consistent segmentation for optical flow estimation, Proc. IEEE Int. Conf. Computer Vision, pp. 1308–1315, 2005.

[16] V. KOLMOGOROV, R. ZABIH, Computing visual correspondence with occlusions using graph cuts, Proc, IEEE Int. Conf. Computer Vision, pp. 508–515, 2001.

[17] J. SUN, Y. LI, S. KANG, H. SHUM, Symmetric stereo matching for occlusion handling, Proc. IEEE Conf. Computer Vision and Pattern Recognition 2, pp. 399–406, 2005.

[18] A. W. FITZGIBBON, Y. WEXLER, A. ZISSERMAN, Image-based rendering using image-based priors, Proc. IEEE Int. Conf. Computer Vision, pp. 1176–1183, 2003.

[19] H. WANG, R. RASKAR, N. AHUJA, Seamless video editing, Int. Conf. on Pattern Recognition, vol. 3, pp. 858–861, 2004.

[20] J. JIA, C. TANG, Image stitching using structure deformation, IEEE Trans. Pattern Analysis and Machine Intelligence 30, 4, pp. 617–631, 2008.

[21] M. Sonka, V. Hlavac, R. Boyle, Image processing, analysis, and machine vision, second edition, Brooks/Cole Publishing Company, 1999.

7. Appendix (Source Code)

```
/** Function getExpShift(const IplImage * srcImage1, const IplImage * srcImage2, int
shiftBits, int shiftRet[])
 *        Find the offsets in x and y direction for the image comparing to the reference
 * Parameters:
 *        srcImage1: the reference frame
 *        srcImage2: the current frame
 *        shiftBits: specified imagepyramid levels
 *        shiftRet[]: array for storing x and y shifts
 */
void getExpShift(const IplImage * srcImage1, const IplImage * srcImage2, int shiftBits,
int shiftRet[]){
    int minErr;
    int curShift[2];

    IplImage *tb1 = cvCreateImage(cvGetSize(srcImage1), 8, 1);
    IplImage *tb2 = cvCreateImage(cvGetSize(srcImage2), 8, 1);
    IplImage *eb1 = cvCreateImage(cvGetSize(srcImage1), 8, 1);
    IplImage *eb2 = cvCreateImage(cvGetSize(srcImage2), 8, 1);

    int i, j;
    if(shiftBits > 0){
        IplImage *sm1Img1 = cvCreateImage(cvSize(srcImage1->width/2, srcImage1-
>height/2), srcImage1->depth, srcImage1->nChannels);
        IplImage *sm1Img2 = cvCreateImage(cvSize(srcImage2->width/2, srcImage2-
>height/2), srcImage2->depth, srcImage2->nChannels);
        //image pyramid
        shrinkImage(srcImage1, &sm1Img1);
        shrinkImage(srcImage2, &sm1Img2);
        //recursive call
        getExpShift(sm1Img1, sm1Img2, shiftBits - 1, curShift);

        cvReleaseImage(&sm1Img1);
        cvReleaseImage(&sm1Img2);
        //re - calculate positions
        shiftRet[0] = curShift[0] * 2;
        shiftRet[1] = curShift[1] * 2;
    } else {
        curShift[0] = curShift[1] = 0;
        //Segmentation, tb denotes for segmented images with a threshold, and eb
stands for the exclusion image
        createBitmaps(srcImage1, tb1, eb1);
        createBitmaps(srcImage2, tb2, eb2);

        minErr = srcImage1->width * srcImage1->height;

        for(i = -1; i < 1; i++)
            for(j = -1; j < 1; j++) {
                int xOffset = curShift[0] + i;
                int yOffset = curShift[1] + j;

                IplImage * shiftedTB2 = cvCreateImage(cvGetSize(srcImage2), 8, 1);
                IplImage * shiftedEB2 = cvCreateImage(cvGetSize(srcImage2), 8, 1);
                IplImage * diffBMP = cvCreateImage(cvGetSize(srcImage2), 8, 1);
                int err;
                shiftBitMap(tb2, xOffset, yOffset, shiftedTB2);
                shiftBitMap(eb2, xOffset, yOffset, shiftedEB2);
                //Compare for the difference for the two images using an XPR operator
                xorBitMap(tb1, shiftedTB2, diffBMP);

                err = totalOneInBitMap(diffBMP);

                if(err < minErr) {
                    shiftRet[0] = xOffset;
                    shiftRet[1] = yOffset;
                    minErr = err;
                }
                cvReleaseImage(&shiftedTB2);
                cvReleaseImage(&shiftedEB2);
            }
            cvReleaseImage(&tb1);
            cvReleaseImage(&tb2);
            cvReleaseImage(&eb1);
            cvReleaseImage(&eb2);
    }
}
/** Function getFileNames(string path, vector<string>& files)
 *          To search in a directory, including all files, typically image files in this case, and
store their names in a vector
 *     Parameters:
 *          path: the directory path to be searched
 *          files: the vector that used to store file names with directory path
 */
void getFileNames(string path, vector<string>& entireNames, vector<string>&
fileNames){
  //File handle
  long   hFile   =   0;
  //File information
  struct _finddata_t fileinfo;
  string p;

  if((hFile = _findfirst(p.assign(path).append("/*").c_str(),&fileinfo)) != -1){
    do{
      //If it is a directory, go in iteration, otherwise, add to list
      if((fileinfo.attrib & _A_SUBDIR)){
        if(strcmp(fileinfo.name,".") != 0 && strcmp(fileinfo.name,"..") != 0)
          getFileNames( p.assign(path).append("/").append(fileinfo.name),
entireNames, fileNames);
      }
      else{
        entireNames.push_back(p.assign(path).append("/").append(fileinfo.name));
                fileNames.push_back(fileinfo.name);
      }
    }  while(_findnext(hFile, &fileinfo) == 0);

    _findclose(hFile);
  }
}
/** Function getBlockShift(IplImage * referenceFrame, IplImage * currentFrame,
BlockMotionVectorPtr *headPtr, BlockMotionVectorPtr *tailPtr)
 *          Main function of block matching algorithm that are used in motion estimation,
block size is preprocessored, as well as search range
 *          The final motion vector is stored as a node which is then insert to a queue
 *  Parameters:
 *          *referenceFrame: reference frame pointer, which points to the reference image,
the previous one.
```

```
 *        *currentFrame: current frame pointer, which points to the second image where
motion has occured in contrast to the reference frame
 *        *headPtr: head node address of the queue
 *        *tailPtr: tail node address of the queue
 */
void getBlockShift(IplImage * referenceFrame, IplImage * currentFrame,
BlockMotionVectorPtr *headPtr, BlockMotionVectorPtr *tailPtr){
    int imageWidth = referenceFrame->width;
    int imageHeight = referenceFrame->height;
    int imageIndexX, imageIndexY;
    int rangeShiftX, rangeShiftY;
    int blockIndexX, blockIndexY;
    double smallestSquareSum = 10000.0000;
    int smallestShifts[2];
    double squareSumMatrix[SEARCH_RANGE * SEARCH_RANGE] = {10000.0000};
    int matrixIndex = 0;
    //calculation loop, adjust here for pixelwise or blockwise
    for(imageIndexX = 0; imageIndexX < imageWidth - BLOCK_SIZE;
imageIndexX+=BLOCK_SIZE)
        for(imageIndexY = 0; imageIndexY < imageHeight - BLOCK_SIZE;
imageIndexY+=BLOCK_SIZE){
            //smallestSquareSum = 10000.0000;
            smallestShifts[0] = 0;
            smallestShifts[1] = 0;

            int isFirstSumInRange = 1;
            //Second loop, in the search range for motion blocks
            for(rangeShiftX = -SEARCH_RANGE; rangeShiftX < SEARCH_RANGE;
rangeShiftX++)
                for(rangeShiftY = -SEARCH_RANGE; rangeShiftY < SEARCH_RANGE;
rangeShiftY++){
                    int isThePositionOutOfRange = 0;
                    double sum = 0;
                    //This loop is to compare pixels in motion blocks
                    for(blockIndexX = imageIndexX; blockIndexX < imageIndexX +
BLOCK_SIZE; blockIndexX++)
                        for(blockIndexY = imageIndexY; blockIndexY < imageIndexY +
BLOCK_SIZE; blockIndexY++){

                            //Control the calculated point inside the image range
                            if(blockIndexX < imageWidth && blockIndexY <
imageHeight && (blockIndexX + rangeShiftX) >= 0 && (blockIndexX + rangeShiftX) <
imageWidth && (blockIndexY + rangeShiftY) >= 0 && (blockIndexY + rangeShiftY) <
imageHeight){
                                int grayDiff = getPixel(currentFrame, blockIndexX,
blockIndexY) - getPixel(referenceFrame, blockIndexX + rangeShiftX, blockIndexY +
rangeShiftY);
                                sum += grayDiff * grayDiff;
                            } else{
                                isThePositionOutOfRange = 1;
                            }
                        }
                    //Out - of - bound control
                    if(!isThePositionOutOfRange){
                        if(isFirstSumInRange){
                            smallestSquareSum = sum;
                            smallestShifts[0] = rangeShiftX;
                            smallestShifts[1] = rangeShiftY;
                            isFirstSumInRange = 0;
                        } else{
                            if(sum <= smallestSquareSum){
                                smallestSquareSum = sum;
                                smallestShifts[0] = rangeShiftX;
                                smallestShifts[1] = rangeShiftY;
                            }
                        }
                    }
                }
                enqueueMotionVectors(headPtr, tailPtr, imageIndexX, imageIndexY,
smallestShifts[0], smallestShifts[1]);
        }
}
/** Function computeDistanceThreshold(BlockMotionVectorPtr
motionVectorQueueNode)
 *          First version: to calculate the mean value of all distance, which is the simplest
method to find a distance threshold
 */
double computeDistanceThreshold(BlockMotionVectorPtr motionVectorQueueNode){
    long count = 0;
    double sum = 0;

    if(motionVectorQueueNode == NULL){
        fprintf(stderr, "Error: Motion Vector Queue is empty\n");
        exit(-1);
    } else{
        while(motionVectorQueueNode != NULL){
            // Computation is limited for non - zero vectors to enhance the motion
distance threshold
            if(motionVectorQueueNode->shiftX != 0 || motionVectorQueueNode-
>shiftY !=0){
                count++;
                sum += sqrt(square(motionVectorQueueNode->shiftX) +
square(motionVectorQueueNode->shiftY));
            }
            motionVectorQueueNode = motionVectorQueueNode->nextPtr;
        }

        if(sum != 0){
            return sum / count;
        } else {
            return 0;
        }
    }
}
/**Function findReferencePoint(ImageInfoNodePtr *headImageInfoNodePtr,
ImageInfoNodePtr *tailImageInfoNodePtr, vector<string>& srcFileNames)
 *          Try to find the best registration point inorder to do tha fastest registration
 * Parameters:
 *          srcFileNames: the vector that stores file names in the folder which contains all
files in image sequence
 * Return:
 *          reference number, an integer that denotes the image index which will be select
as reference.
 */
int findReferencePoint(ImageInfoNodePtr *headImageInfoNodePtr, ImageInfoNodePtr
*tailImageInfoNodePtr, vector<string>& srcFileNames){
    long referencePoint = 0, index = 1, refIndex = 0, matchNumbers = 0;
    int groupChanged = 0;

    vector<string>::iterator iter;

    while(index < (long) srcFileNames.size()){
        IplImage *src1 = NULL, *src2 = NULL,
            *grayPlane1 = NULL, *grayPlane2 = NULL,
            *shunkedGrayPlane1 = NULL, *shunkedGrayPlane2 = NULL;
        BlockMotionVectorPtr headMotionVectorPtr = NULL;
        BlockMotionVectorPtr tailMotionVectorPtr = NULL;
        DistanceFrequencyNodePtr headDistanceFrequencyPtr = NULL;
        DistanceFrequencyNodePtr tailDistanceFrequencyPtr = NULL;

        if(groupChanged)
            matchNumbers = groupChanged = 0;
        // Read image ( same size, same type )
        src1 = cvLoadImage(srcFileNames[refIndex].c_str(), 1);
        src2 = cvLoadImage(srcFileNames[index].c_str(), 1);
        // File check
        if(src1 ==  NULL) {
            fprintf(stderr, "Error: Error loading src1!\n");
            exit(-1);
        }
        if(src2 ==  NULL) {
            fprintf(stderr, "Error: Error loading src2!\n");
            exit(-1);
        }
        //create gray planes
        allocateOnDemand(&grayPlane1, cvGetSize(src1), src1->depth, 1);
        allocateOnDemand(&grayPlane2, cvGetSize(src2), src2->depth, 1);
        cvCvtColor(src1, grayPlane1, CV_BGR2GRAY);
        cvCvtColor(src2, grayPlane2, CV_BGR2GRAY);
        //shrink images according to pyramid level
        shrinkImage(grayPlane1, &shunkedGrayPlane1, PYRAMID_LEVEL);
        shrinkImage(grayPlane2, &shunkedGrayPlane2, PYRAMID_LEVEL);
        //Find a series of motion vectors for blocks in the current frame
        getBlockShift(shunkedGrayPlane1, shunkedGrayPlane2,
&headMotionVectorPtr, &tailMotionVectorPtr);

        int shiftArray[2] = {0};
        //Find the shift array that stores the x and y shift of the current frame, which
will be shifted to do the registration
        getImageShift(headMotionVectorPtr, &headDistanceFrequencyPtr,
&tailDistanceFrequencyPtr, shiftArray);

        if(shiftArray[0] == 0 && shiftArray[1] == 0){
            //Two images match, move to next image, continue comparison
            matchNumbers++;
            index++;
        } else{
            //COMPUTE MOTION VECTOR, ENQUEUE
            enqueueImageInfo(headImageInfoNodePtr, tailImageInfoNodePtr, refIndex,
matchNumbers, shiftArray[0], shiftArray[1]);

            refIndex = index;
            index++;
            groupChanged = 1;
        }
        free(headMotionVectorPtr);
        free(tailMotionVectorPtr);
        free(headDistanceFrequencyPtr);
        free(tailDistanceFrequencyPtr);
        cvReleaseImage(&src1);
        cvReleaseImage(&src2);
        cvReleaseImage(&grayPlane1);
        cvReleaseImage(&grayPlane2);
        cvReleaseImage(&shunkedGrayPlane1);
        cvReleaseImage(&shunkedGrayPlane2);
    }
    //Enqueue the nodes that stores shifting information in the image sequence
    enqueueImageInfo(headImageInfoNodePtr, tailImageInfoNodePtr, refIndex,
matchNumbers, 0, 0);
    ImageInfoNodePtr tempPtr = NULL;
    tempPtr = (ImageInfoNodePtr)malloc(sizeof(ImageInfoNode));

    if(tempPtr != NULL){
        tempPtr = *headImageInfoNodePtr;
        int largestMatchNumbers = tempPtr->matchedImageNumbers;
        tempPtr = tempPtr->nextPtr;

        while(tempPtr != NULL){
            if(tempPtr->matchedImageNumbers > largestMatchNumbers){
                largestMatchNumbers = tempPtr->matchedImageNumbers;
                referencePoint = tempPtr->imageIndex;
            }
            tempPtr = tempPtr->nextPtr;
        }
    } else {
        fprintf(stderr, "Error: New Image Info vector is not created, Out of
memory?\n");
        exit(-1);
    }
    free(tempPtr);
    return referencePoint;
}
/** Function computeBackgroundMotion(DistanceFrequencyNodePtr
distanceFrequencyQueueNode, int shifts[])
 *      Find background motion by frequency, result stored in shifts[]
 */
void computeBackgroundMotion(DistanceFrequencyNodePtr
distanceFrequencyQueueNode, int shifts[]){
    long maxFrequency = distanceFrequencyQueueNode->frequency;
    int shiftX = distanceFrequencyQueueNode->shiftX;
    int shiftY = distanceFrequencyQueueNode->shiftY;

    if(distanceFrequencyQueueNode == NULL){
        fprintf(stderr, "Error: Motion Vector Queue is empty\n");
        exit(-1);
    } else{
        while(distanceFrequencyQueueNode != NULL){
            if(distanceFrequencyQueueNode->frequency > maxFrequency){
                shiftX = distanceFrequencyQueueNode->shiftX;
                shiftY = distanceFrequencyQueueNode->shiftY;
            } else if(distanceFrequencyQueueNode->frequency == maxFrequency){
                shiftX = (shiftX + distanceFrequencyQueueNode->shiftX) / 2;
                shiftY = (shiftY + distanceFrequencyQueueNode->shiftY) / 2;
            }

            distanceFrequencyQueueNode = distanceFrequencyQueueNode->nextPtr;
        }
        *(shifts) = (int)shiftX;
        *(shifts + 1) = (int)shiftY;
    }
}

/** Function getImageShift(BlockMotionVectorPtr motionVectorQueueNode,
DistanceFrequencyNodePtr *headPtr, DistanceFrequencyNodePtr *tailPtr, int shiftCo[])
 *      Find the shift array that stores the x and y shift of the current frame, which will
be shifted to do the registration
 */
void getImageShift(BlockMotionVectorPtr motionVectorQueueNode,
DistanceFrequencyNodePtr *headPtr, DistanceFrequencyNodePtr *tailPtr, int
shiftCo[]){

    double sum = 0;
   double distanceThreshold = computeDistanceThreshold(motionVectorQueueNode);

    printf("%f\n", distanceThreshold);

    if(motionVectorQueueNode == NULL){
        fprintf(stderr, "Error: Motion Vector Queue is empty\n");
        exit(-1);
    } else{
        while(motionVectorQueueNode != NULL){
            if(sqrt(square(motionVectorQueueNode->shiftX) +
square(motionVectorQueueNode->shiftY)) < distanceThreshold){
                enqueueDistanceFrequency(headPtr, tailPtr, motionVectorQueueNode-
>shiftX, motionVectorQueueNode->shiftY);
            }
            motionVectorQueueNode = motionVectorQueueNode->nextPtr;
        }

        if(distanceThreshold != 0){
            //Find background motion by frequency, result stored in shifts[]
            computeBackgroundMotion(*headPtr, shiftCo);
        } else{
            *shiftCo = 0;
            *(shiftCo + 1) = 0;
        }
        *shiftCo = (*shiftCo) * pow(2.0, PYRAMID_LEVEL);
        *(shiftCo + 1) = *(shiftCo + 1) * pow(2.0, PYRAMID_LEVEL);
    }
}
/** Function shiftImageSequence(ImageInfoNodePtr imageInfoNode, vector<string>&
srcFileNames, vector<string>& dstFileNames)
 *      Shift the entire sequence one by one, and save to new files
 * Parameters:
 *      imageInfoNode: the queue node that stores shift information of the image
 *      srcFileNames: the vector that includes all source image file names
 *      dstFileNames: the vector that includes all new image file names where they
could be saved
 */
```

```
void shiftImageSequence(ImageInfoNodePtr imageInfoNode, vector<string>&
srcFileNames, vector<string>& dstFileNames){
    long matchedNumber = 0;
    long index = 0;

    while(imageInfoNode != NULL){

        IplImage  * srcImage = cvLoadImage(srcFileNames[index].c_str(),1);

        IplImage  * shiftedImage = cvCreateImage(cvGetSize(srcImage), srcImage-
>depth, srcImage->nChannels);
        cvSet(shiftedImage, CV_RGB(255, 255, 255), NULL);

        shiftImage(srcImage, imageInfoNode->shiftX, imageInfoNode->shiftY,
shiftedImage);
        printf("Image: %d, aligned at (%d, %d)!\n", index,  imageInfoNode->shiftX,
imageInfoNode->shiftY);

        cvSaveImage(dstFileNames[index].c_str(), shiftedImage, 0);
        index++;

        if(imageInfoNode->matchedImageNumbers == 0){
            imageInfoNode = imageInfoNode->nextPtr;
        } else{
            matchedNumber++;

            if(matchedNumber > imageInfoNode->matchedImageNumbers){
                matchedNumber = 0;
                imageInfoNode = imageInfoNode->nextPtr;
            }
        }
    }
}
/** Function createNewFrame(IplImage ** newFrame, IplImage *refFrame, IplImage
*curFrame, BlockMotionVectorPtr motionVectorQueueNode, double threshold)
 *          Main function of creating the new image between frames
 */
void createNewFrame(IplImage ** newFrame, IplImage *refFrame, IplImage
*curFrame, BlockMotionVectorPtr motionVectorQueueNode, double threshold){
    int motions[2];
    //select the most probable motions
    adjustMotionBlock(motionVectorQueueNode, threshold, motions);

    if(motionVectorQueueNode == NULL){
        fprintf(stderr, "Error: Motion Vector Queue is empty\n");
        exit(-1);
    } else{
        while(motionVectorQueueNode != NULL){
            if(shiftedDistance(motionVectorQueueNode->shiftX,
motionVectorQueueNode->shiftY) >= threshold){
                fillMotionBlock(newFrame, curFrame, motionVectorQueueNode->x *
(pow(2.0, PYRAMIDLEVEL)), motionVectorQueueNode->y * (pow(2.0,
PYRAMIDLEVEL)), (*motions) * (pow(2.0, PYRAMIDLEVEL)), (*(motions + 1)) *
(pow(2.0, PYRAMIDLEVEL)));
            } else{
                fillBackGroundBlock(newFrame, refFrame, curFrame,
motionVectorQueueNode->x * (pow(2.0, PYRAMIDLEVEL)), motionVectorQueueNode-
>y * (pow(2.0, PYRAMIDLEVEL)));
```

```
                }
                motionVectorQueueNode = motionVectorQueueNode->nextPtr;
            }
        }
}
/** Function void adjustMotionBlock(BlockMotionVectorPtr *motionVectorQueueNode)
 *         Find the point that needs interpolation and then create a new image
 */
void adjustMotionBlock(BlockMotionVectorPtr motionVectorQueueNode, double
distanceThreshold, int motions[]){
        DistanceFrequencyNodePtr headDistanceFrequencyPtr = NULL;
        DistanceFrequencyNodePtr tailDistanceFrequencyPtr = NULL;
        BlockMotionVectorPtr headMotionVector = motionVectorQueueNode;

        while(motionVectorQueueNode != NULL){
            if(sqrt(square(motionVectorQueueNode->shiftX) +
square(motionVectorQueueNode->shiftY)) >= distanceThreshold){
                enqueueDistanceFrequency(&headDistanceFrequencyPtr,
&tailDistanceFrequencyPtr, motionVectorQueueNode->shiftX,
motionVectorQueueNode->shiftY);
            }
            motionVectorQueueNode = motionVectorQueueNode->nextPtr;
        }

        long largestFrequency = headDistanceFrequencyPtr->frequency;
        int likelyMotionX = headDistanceFrequencyPtr->shiftX;
        int likelyMotionY = headDistanceFrequencyPtr->shiftY;

        while(headDistanceFrequencyPtr != NULL){
            if(largestFrequency < headDistanceFrequencyPtr->frequency){
                largestFrequency = headDistanceFrequencyPtr->frequency;
                likelyMotionX = headDistanceFrequencyPtr->shiftX;
                likelyMotionY = headDistanceFrequencyPtr->shiftY;
            }
            headDistanceFrequencyPtr =  headDistanceFrequencyPtr->nextPtr;
        }
        *motions = likelyMotionX;
        *(motions + 1) = likelyMotionY;
}
/**Implementation of Function interpolateInSequence(BlockMotionVectorPtr
motionVectorQueueNode, double interpolateThreshold)
 */
void interpolateInSequence(char * directory, vector<string>& fileNames,
vector<string>& srcFileNames, double *thresholds, int quantity){
        //Find maximum threshold
        double maxThreshold = *(thresholds);

        for(int i = 1; i < quantity; i++){
            if(*(thresholds + i) > maxThreshold){
                maxThreshold = *(thresholds + i);
            }
        }

        for(long i = 0; i < quantity; i++){
            if(*(thresholds + i) >= maxThreshold * INTERPOLATION_RANGE){
                long index = i;
                //Read images
                IplImage * refFrame = cvLoadImage(srcFileNames[index].c_str());
```

```
			IplImage * curFrame = cvLoadImage(srcFileNames[index + 1].c_str());

			BlockMotionVectorPtr headMotionVectorPtr = NULL;
			BlockMotionVectorPtr tailMotionVectorPtr = NULL;

			IplImage * refGrayPlane = NULL, * curGrayPlane = NULL,
				* shrunkRefPlane = NULL, * shrunkCurPlane = NULL, *
interFrame = NULL, * newFrame = NULL;

			allocateOnDemand(&interFrame, cvGetSize(curFrame), curFrame->depth,
curFrame->nChannels);
			allocateOnDemand(&newFrame, cvGetSize(curFrame), curFrame->depth,
curFrame->nChannels);
			//create grey planes
			allocateOnDemand(&refGrayPlane, cvGetSize(refFrame), IPL_DEPTH_8U,
1);
			cvCvtColor(refFrame, refGrayPlane, CV_BGR2GRAY);
			allocateOnDemand(&curGrayPlane, cvGetSize(curFrame), IPL_DEPTH_8U,
1);
			cvCvtColor(curFrame, curGrayPlane, CV_BGR2GRAY);
			//create image pyramid
			shrinkImage(refGrayPlane, &shrunkRefPlane, PYRAMIDLEVEL);
			shrinkImage(curGrayPlane, &shrunkCurPlane, PYRAMIDLEVEL);
			//Do with pyramid to speed up
			getBlockShift(shrunkRefPlane, shrunkCurPlane, &headMotionVectorPtr,
&tailMotionVectorPtr);

			double distanceThreshold =
computeDistanceThreshold(headMotionVectorPtr);

			//Interpolation
			createNewFrame(&interFrame, refFrame, curFrame, headMotionVectorPtr,
distanceThreshold);
			fillBlankPixels(curFrame, interFrame, &newFrame);

			printf("New frame between image %ld and %ld interpolated!\n", index,
index + 1);
			//new image re - naming
			char newName[500];
			strcpy(newName, directory);
			strcat(newName, "/inter_");
			strcat(newName, fileNames[index].c_str());

			cvSaveImage(newName, newFrame);
			//Release memories
			cvReleaseImage(&refFrame);
			cvReleaseImage(&curFrame);
			cvReleaseImage(&refGrayPlane);
			cvReleaseImage(&curGrayPlane);
			cvReleaseImage(&shrunkRefPlane);
			cvReleaseImage(&shrunkCurPlane);
			cvReleaseImage(&interFrame);
			cvReleaseImage(&newFrame);
		}
	}
}
```