University College Dublin

Bachelors of Electronic Engineering

Real Time Motion Estimation

April 26^{th}, 2019

Author:

James Carron

Supervisor:

Dr. John T. Sheridan

Abstract

With the explosion of automation real-time motion estimation is an expanding field, allowing the huge streams of information to be compressed into simple to interpret data representing the motion of a camera relative to its surroundings.

Inferring physical motion from digital images is difficult task for computers to complete. Current methods allow accurate results, however, they are computationally expensive to apply and they necessitate a compromise between spatial and temporal resolution to achieve close to real time operation.

In this project a novel method developed in University College Dublin (UCD) demonstrating much reduced computational expense, robustness to noise and error detection is demonstrated which could enable real time operation with high temporal and spatial resolution. Methods of improving the throughput of the method are investigated and tested.

Performance of the proposed method is investigated on varying levels of hardware including low power, low cost hardware previously considered unusable for this task.

Acknowledgements

First of all, I would like to acknowledge the guidance and advice of my supervisor, Prof. John T. Sheridan, throughout this project. His support throughout this project was exemplary and lead to a very logical and real-world approach to tackling the problems presented.

To my fellow researchers, Mark Coughlan, Cillian Cooke and Kevin O'Flannagain. I would also like to extend my gratitude for their assistance in getting to grips with this brand new field of engineering, for their advice on improving this thesis, and for their willingness to give time from their own research. To Min Wan, I would like to extend my gratitude, for her help in the laboratory. My classmates also deserve to be recognised, because it is without doubt that I would not have made it through this degree without them.

I would also like to thank Melanie Ng Tung Hing and Conor Foy for their contributions to this project. Melanie your ability to turn a project report into a professional document is outstanding and transformed my research into the document presented. Conor the way you are able to logically work through problems outside of your field of expertise is astounding and helped me overcome many mental blocks and tunnel vision induced ruts I got stuck in. An honourable mention to Josh King and George Ridgway without whom I would have desecrated the paradigm of Θ Notation.

Last, but not least, I would like to thank my siblings and parents for putting up with all the "Engineering is so hard" moaning and for all the support and laughs over the years. I will be forever grateful for everything you have done for me to get me this far.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern Digital Image sensors are remarkable feats of engineering which allow us to capture and quantify the world around us in amazing detail but the intricate operation of their sensors is abstracted away from the user. The methods by which they capture this information is imperative to understanding of how we can use this date to make inferences about the world around us. This section will introduce the operation of and technical language used to describe real time digital image motion estimation.

1.1 Operation of an Image Sensor

An image sensor converts electromagnetic waves incident on the sensor into electrical signals over discrete area's (pixels) using light sensors. Modern sensors are capable of capturing millions of pixels of information several times per second.

1.1.1 Sensing Incident Light

Light sensors utilise a process called photoconversion [1] whereby a photon incident on the surface of a semiconductor creates an electron-hole pair where it absorbs to measure the intensity of light incident on a pixel. This causes a current to flow through the image sensor which creates a voltage which can be sampled.

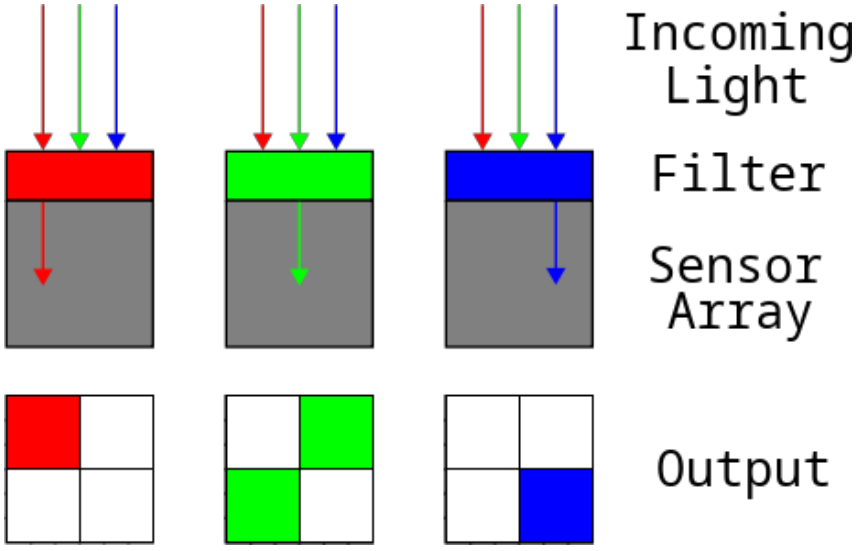


Figure 1.1: Composition of an Image Sensor

Thus, the light intensity can be quantified using the voltage output. The voltage output is based on the total current in the photodiode which is determined by the amount of photons that are incident on the sensor.

$$V_{out} = V_{photodiode} - \frac{i_{photo}}{C_D} \tag{1.1}$$

Where V_{out} is the output voltage measured across the photodiode, i_{photo} is the photocurrent and C_D is the capacitance of the photodiode. Thus to quantify how much light hits the sensor, the voltage output can be directly measured. The currents produced by the photoconversion is typically minute, on the order of femtoamperes (10^{-15} A). As the capacitance of the photodiode is also generally in the femtofarad range, this offsets the small photocurrent values.

Modern light sensors are produced using silicon which has a bandgap of 1.1 eV which is perfect for capturing light in the visible and near infrared spectrum. If a photon hits silicon, and the photon has an energy larger than 1.1 eV, then it will be absorbed into the silicon and produce charge. [1].

The energy of a photon is defined as:

[1] subject to the quantum efficiency of silicon at that wavelength.

$$E = h^c/\lambda \tag{1.2}$$

Where E is the energy of the photon, h is Plank's constant, c is the speed of light and λ is the wavelength of the light. The visible light spectrum contains wavelengths from 450 nm to 650 nm, which corresponds to photon energies of 2.75 eV and 1.9 eV respectively.

1.1.2 Sensitivity of a Light Sensor

There are multiple factors which effect the sensitivity (minimum light intensity detectable) of a light sensor. The larger the sensor area can collect more photons, and therefore is more sensitive to lower light intensities with the same voltage sensitivity. The minimum voltage detectable by the sampling circuitry. Using different sensor architectures (Charged Couple Diode or Complimentary Metal Oxide Semiconductor) also yields different results however this topic is outside the scope of this thesis as it has little relevance to the investigation of the proposed method. Exposing the image for a longer duration allows the light sensor to collect more photons before the voltage is sampled and reset.

1.1.3 Representing Pixel Values

While bit depth is also used to refer to the number of bits used to represent a sample in an audio recording for the purposes of this thesis bit depth (Colour bit depth) will be used to refer to the number of bits used to represent the colour of a pixel in an image.

Red, green and blue (RGB) are commonly used to represent colour on electronic devices and are the basis of the additive[2] RGB colour model.

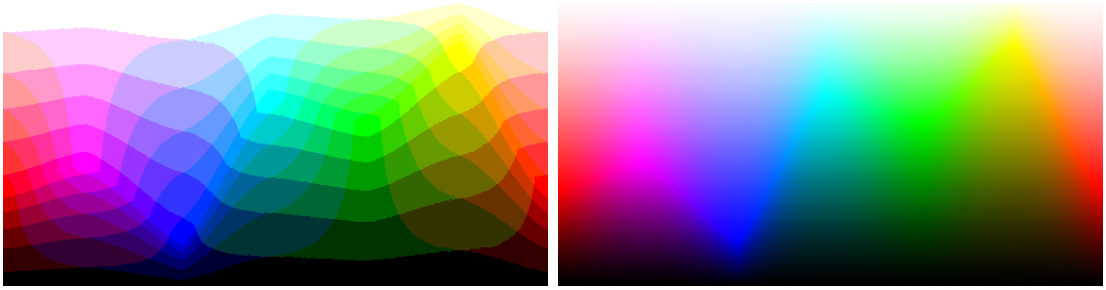
In the RGB system, the red, green and blue dots are assigned brightness values along some scale, for example using an 8-bit depth we get the range [0, 255], where 0 is dark and 255 is bright. By specifying the three values red, green and blue (also known as primaries

[2]Additive colours get lighter when mixed.

or channels), you can specify the exact colour that will be displayed. Hence by increasing the number of bits we use to represent a pixel we can vary the number of colours we can represent. The number of colours possible can be described as:

$$\text{Number of Possible Colours } = 2^{R_b} \times 2^{G_b} \times 2^{B_b} = 2^{R_b+G_b+B_b} \tag{1.3}$$

Where R_b, G_b, B_b represent the number of bits to represent the R,G & B Channels respectively.



(a) 8 Bit Colour Depth (b) 24 Bit Colour Depth

Figure 1.2: Effect of bit depth on kept information [2].

The choice of colours is related to the physiology of the human eye; good primaries are stimuli that maximize the responses of the human eye to light of different wavelengths[3].

Name	Number of Colours Possible	Bits per pixel	Bits per channel (R/G/B)
8-bit colour	256	8	3/3/2[4]
High colour	65536	16	5/6/5
18-bit	262144	18	6/6/6
True colour	24	16,777,216	8/8/8
Deep colour	30/36/48	10/12/16 respectively	

Table 1.1: Common RGB colour bit depths [5]

[3]The three kinds of light-sensitive photoreceptor cells in the human eye (cone cells) have peak response to long wavelengths (570 nm, yellow), medium wavelengths (540 nm, green), and short wavelengths (440 nm, violet) light respectively [3].

In the scope of this thesis using three channels to represent each pixel provides no advantage and simply makes the notation more cumbersome. Henceforth we will consider each pixel to be represented by a single intensity value (greyscale) encoded into b bits.

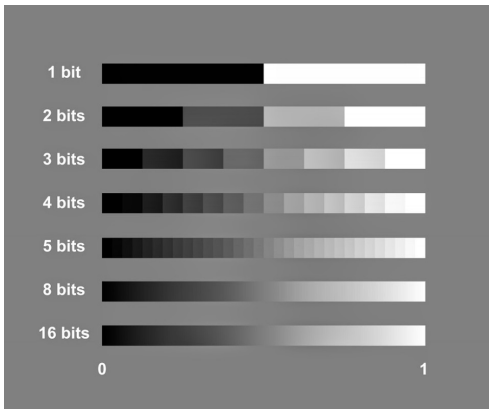


Figure 1.3: Grayscale values available to a sensor with different bit depths [6]

In sampling the light intensity the continuous value is mapped to the nearest bit level value. Where 0 indicates no incident light, and 1 indicates full saturation of the sensor in steps of $1/(2^b - 1)$. Figure 1.4 demonstrates light being quantised into a 3 bit value which enables steps of $1/(2^3 - 1) \approx 0.14$.

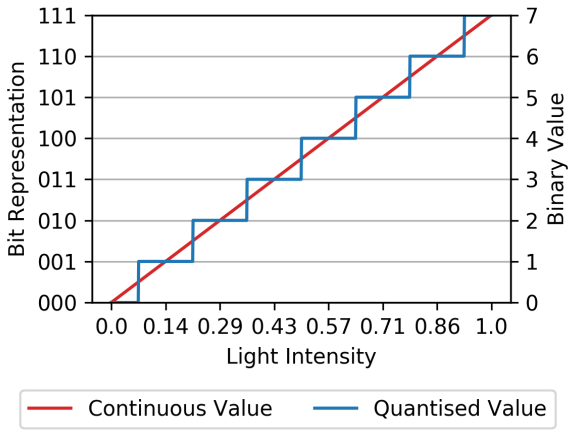
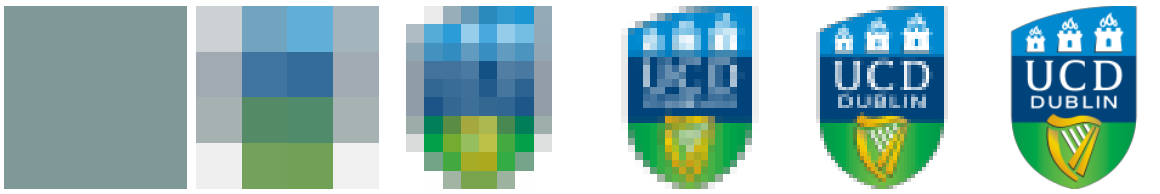


Figure 1.4: Quantisation of light intensity to the nearest bit level using three bits.

1.2 Resolution

Resolution in the scope of an imaging system is separated into spatial and temporal resolution. Spatial resolution is defined as the number of pixels that an image sensor has that contributes to the final output. Spatial resolution is often defined in terms of the number of rows (N) and columns (M) a sensor of pixels has or the total number of pixels ($N \times M$).



(a) 1×1 pixels (b) 4×4 (c) 10×10 (d) 25×25 (e) 50×50 (f) 150×150

Figure 1.5: Effect of spatial resolution on feature differentiation.

The common digital image sensor resolutions that will be considered in this thesis are:

Name	Resolution
Video Graphics Array (VGA)	640×480
Super Video Graphics Array (SVGA)	800×600
High-definition (HD)	1280×720
High-definition Plus (HD+)	1600×900
Full High-definition (FHD)	1920×1080
4K UHD	3840×2160
8K UHD	7680×4320

Table 1.2: Common display resolutions

Temporal resolution refers to the precision of a measurement with respect to time. In the context of imaging this refers to the number of images taken per second (frame rate).

With most imaging systems there is a trade off between spatial and temporal resolution as there is usually a hard limit on how much information per second the hardware can process and store. The current fastest 4Mpx camera on the market, the Phantom v2640 has a max throughput of 26 gigapixels per second[7]. Hence by lowering the resolution higher frame rates can be achieved.

Resolution	Max Frame rate
2048 × 1952	6,600
2048 × 1440	8,800
1920 × 1080	12,500
1024 × 976	14,740
1792 × 720	16,690
640 × 480	28,760
1792 × 8	303,460

Table 1.3: Phantom x2650 resolution vs frame rate trade off.

Motion estimation methods also have this limitation which is tied to the number of operations it takes to compute a motion estimation for a given image. And the number of operations per second, that the compute device the algorithm is run on, can execute.

Chapter 2

Background

Inferring physical motion from digital images is difficult task for computers to complete. If we can quantify the movement between two images, knowing the rest of the image system (lensing etc) we can quantify the physical movement that occurred between the two images. With the explosion of automation this data is required for real time applications and current methods cannot process the high spatial resolution images available with Modern Digital Image sensor technology. To better understand the task at hand we delve into its applications in section 2.1, current solutions in section 2.2 and prior research in this field in section 2.3, to give the reader some context for the need to research the area of digital image motion estimation.

2.1 Applications

General Automation. With the rise of robotics the assimilation of information to drive their decisions has become increasingly relevant. For example, autonomous transportation is required to make hundreds of decisions per second to ensure passengers and cargo are transported to their destination without causing any harm to the public.

Microscopy. In some applications, the moving object is so minute, that an invasive method will effect the accuracy of the result. Attaching a marker to a small moving object can significantly effect its overall mass and aerodynamic drag, hence effecting its motion characteristics[8].

Clinical Application. Often invasive methods of motion estimation are needed to identify patient ailments, however, with the fragility of living specimens this is often not ideal and can effect the accuracy of results.

An example of this is Ocular microtremor (OMT) which is a physiological high-frequency (up to 150 Hz) low-amplitude (25-2500 nm peak-to-peak) involuntary motion of the human eye. Proposed applications include monitoring the depth of anaesthesia of a patient in surgery, prediction of outcome in coma, and diagnosis of brain stem death[9].

Conventionally clinical OMT investigations have used mechanical contact piezoelectric probes or strain gauges which apart from being highly uncomfortable especially for anaesthetized patients who experience blepharospams (spasm of the eye lid), also requires the use of sterile piezoelectric probes, which increases difficulty and cost. Other non-invasive systems employ the use of speckle interferometric sensors.

Far Field Motion Estimation. The measurement of the small amplitude vibrations of large structures, such as buildings and bridges, is possible using invasive methods attached to the target. Using a camera sensor allows far field measurement of these vibrations when access to the site may not be possible or dangerous to humans. For instance in a situation with a damaged structure which is liable to collapse or in the case of hazardous chemical spills or radiation fallout.

Consumer Motion Estimation. The proliferation of high resolution image sensors in modern consumer electronics means accurate motion estimation can be made more available to the masses as they already have the required equipment.

2.2 Current Digital Image Motion Detection Algorithms

To understand the advantages and limitations of the proposed method it is imperative to have a detailed understanding of the dominant digital image motion detection algorithms. Here the three main categories of digital image motion estimation, correlation, edge detection and object detection will be laid out.

2.2.1 Correlation Motion Estimation

Correlation is the most widely used digital image motion estimation technique. Correlation methods refer to the group of methods that test the similarity of two images using a convolution method, which involves multiplying the translated and test image pixel by pixel. Two dissimilar images (e.x. an original image and its translated equivalent) will return a low similarity value however if a digital translation is applied to the captured image which cancels out the physical translation, a far higher similarity value will be output.

It is defined mathematically as:

$$(f \star g)(\tau) \coloneqq \int_{-\infty}^{+\infty} f^*(t)g(t+\tau)dt \tag{2.1}$$

This works as similar values multiplied by each other will result in a maximum value. However, two dissimilar values will result in a lower overall value. Consider the Pixel Intensity sequence $[0, 1]$. The two bright values multiplied together and two dark values multiplied together e.x. $(0*0 + 1*1 = 1)$ equates to a large value indicating their similarity. However, if this image is inverted and correlated to the original $(0*1 + 1*0 = 0)$ a smaller value is output from the correlation algorithm indicating the reduced similarity. This works as high (bright) similar values compound (n^2) while dissimilar values result in a lower value.

Auto Correlation entails taking the original image and correlating it with itself. This gives us a representation describing the similarity of the image with itself which we will strive to replicate with the test image. This helps account for the fact that if we correlated any original image with a completely bright image we would get an unusually large value which could be misinterpreted as denoting strong similarity between the images.

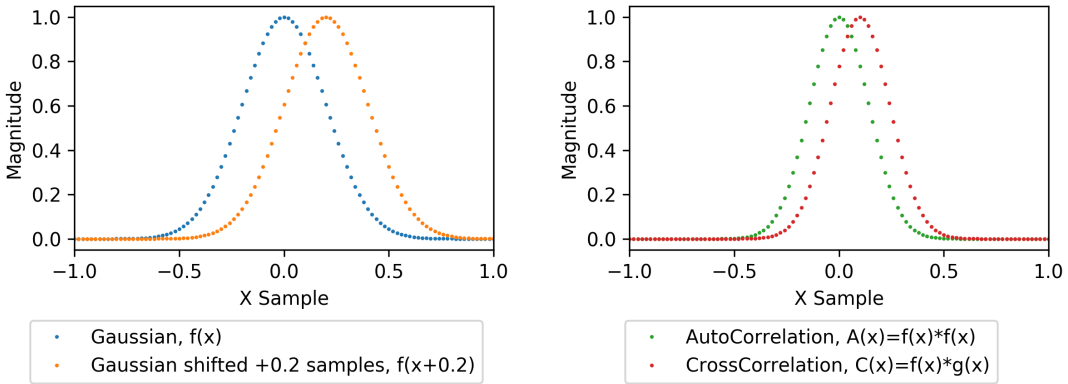
Cross Correlation operates by windowing (cropping a portion of the image) the test image (translated image) and repeatedly applying a digital translation to try cancel out the physical translation. This produces a single value for how similar the test image is to the original image at each position. The digital translation at which the images have the highest similarity quantifies the negative of the translation.

Example

Consider a 1D Gaussian described by the following probability density equation,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)} \tag{2.2}$$

where σ is the standard deviation, μ is the mean and x of the Gaussian.



(a) Gaussian functions f(x), g(x)=f(x+dx). (b) Correlations output, normalised to 1.

Figure 2.1: Applying the correlation algorithm to a Gaussian function, and its shifted equivalent. σ =0.1, $\mu = 0$

Domain	Original Image	Translated Image
Time	$x(n_1, n_2)$	$x(n_1 - m_1, n_2) - m_2$
Frequency	$x(w_1, w_2)$	$e^{-jw_1m_1}e^{-jw_2m_2}x(w_1, w_2)$

Table 2.1: Image and translated equivalent described in time domain and frequency domain notation.

Following this process the distance between the peaks of the correlations is found to be 0.2 samples which is a correct estimate of the 0.2 sample translation.

Frequency Domain Correlation

Frequency Domain Correlation utilises the shifted frequency domain property to reduce the computational expense of calculating the convolutions. Once calculated the translation deltas m_x and m_y can be separated out and converted back to the time domain.

Hence the portion of the expression related to the translation in the frequency domain can be separated out and converted back to the time domain, returning the translation deltas m_1 and m_2.

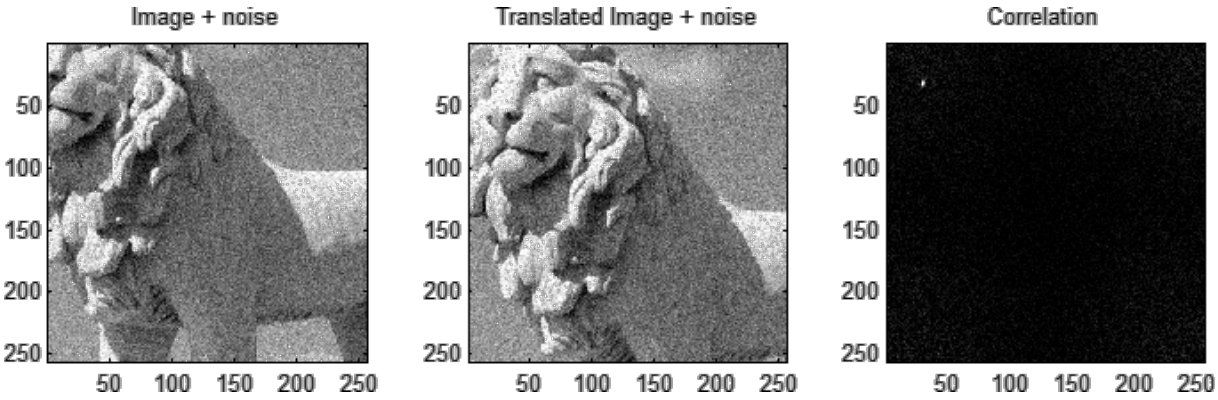


Figure 2.2: Correlation example [10].

Interpolation

Fundamentally Correlation Methods cannot detect subpixel motion. Therefore, they have to super-sample the image to generate a higher spatial resolution image which they then operate on to infer a sub pixel translation.

Interpolation is a method of inserting extra data samples into a function based on the existing data samples. The aim of interpolating the data is to approximate some of the data that is lost when it is sampled by the camera. It is important to note that no extra data is created, it can only be approximated.

The factor by which the resolution is increased is determined by the number of interpolation samples inserted between existing samples. For example, if a sample was inserted halfway between each existing sample, the resolution is doubled. To increase the resolution tenfold 9 samples are inserted between each existing sample.

There are various methods of interpolation to determine the value of the samples inserted. Linear interpolation adds new samples on a linear path between existing samples. In linear interpolation, the inserted sample would be the average of the existing samples before and after the inserted sample. Spline interpolation uses low degree polynomials in order to fit the new samples smoothly with the existing ones. This gives a more accurate approximation of the lost data but is harder to calculate.

(a) Original Image. (b) Image interpolated by a factor of 10.

Figure 2.3: Applying interpolation to an image.

2.2.2 Edge Detection

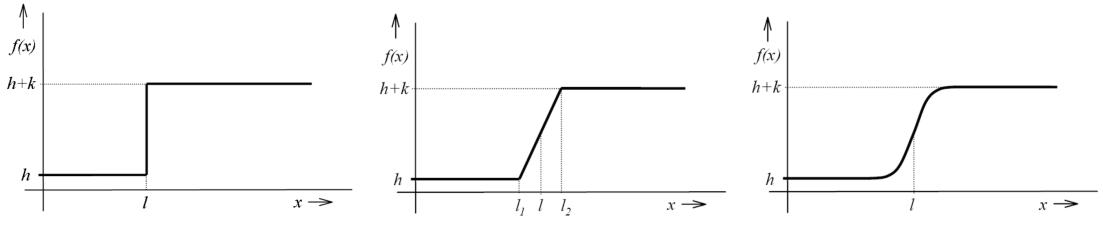
The physical outline is one of the most important properties of an object that machine vision methods rely on to quantify the contents of an image or differences between images. It is often used for detecting translations between images, thus the performance of edge detection methods to identify the outlines of objects directly affects the accuracy of results. Classic pixel-level edge detection can at present only judge the edge position in a pixel but cannot more precisely identify where within that pixel the edge lies.

Subpixel methods propose a solution by subdividing the pixel and thereby more accurately predicting the position of the edge. Often this increased accuracy is associated with increased compute times and reduced robustness.

Edge Models

To detect edge locations first a model of an edge must be defined. The most simple model is a step change function. This model is characterised by three parameters: back-ground intensity h, edge contrast k, and edge location l.

In actuality a ramp edge model is more suitable, as the brightness changes gradually as the camera pixels rarely perfectly align with the object edges, resulting in quantization errors. A ramp edge model has four parameters: background intensity h, edge contrast k, edge beginning l_1 and edge end l_2 . Location of the edge l is equal to the arithmetic average of l_1 and l_2 .



(a) Step edge (b) Ramp edge (c) Blurred edge

Figure 2.4: Common edge models.

Other real world factors we want to account for include defocusing or blurring to due finite optical depth of field. There are a variety of algorithms can be used to approximate this including hyperbolic tangent and Gaussian.

Conventional Edge Detection Methods

Most edge detectors at subpixel level fall into three groups: fitting, interpolation and moment based methods.

Fitting methods use continuous functions such as hyperbolic tangent or B-spline to model an edge sampled by a digital sensor and the image is matched to this. Then the edge location is taken to be the inflection point of the continuous function.

Interpolation based methods achieve subpixel accuracy by interpolating values for subpixels based on values of surrounding pixels using linear and spline fitting methods. Then conventional edge detectors such as Canny[11] and LoG[12] operators are employed to determine the final edge position.

Moment based operators apply statistical moments to determine rates of change within the dataset and establish the parameters associated with the two sides of the edge. Thus, the edge location can be identified as the area at which these two intensities meet.

Operation of Gray Level Moment Operators

The standard gray level moment proposed by Tabatabai and Mitchel[13] operates on a 1-D images as follows. Based on the first three moments m_1, m_2, m_3 of the input data sequence:

$$m_i = \frac{1}{n}\sum_{j=1}^{n} x_i^j \ldots i = 1, 2, 3 \tag{2.3}$$

where $x_1, x_2, ...x_n$ are image samples. Let suppose that they are the samples of ideal step edge and p_h is a number of samples with gray level h (they are the pixels on the left of the edge). If we define the densities p_2 and p_2 as:

$$p_1 = \frac{p_h}{n}, \quad p_2 = \frac{n - p_h}{n} = 1 - p_1 \tag{2.4}$$

then solution of three equations:

$$m_1 = (1-p_2)h + p_2(h+k), \quad m_2 = (1-p_2)h^2 + p_2(h+k)^2, \quad m_3 = (1-p_2)h^3 + p_2(h+k)^3 \tag{2.5}$$

with three unknown variables h, k, p_2 results in:

$$p_2 = {}^1\!/_2(1 - s\sqrt{{}^1\!/_{4+s^2}}), \quad h = m_1 - \beta\sqrt{{}^{p_2}\!/_{p_1}}, \quad k = 2\beta\sqrt{{}^{p_2}\!/_{p_1}} \tag{2.6}$$

where

$$s = \frac{m_3 + 2m_1^3 - 3m_1m_2}{\beta^3}, \quad \beta^2 = m_2 = m_1^2 \tag{2.7}$$

Where an edge is present $p_h = n.p_1$ is not an integer and represents the subpixel edge location.

2.2.3 Object Detection

One of the newest digital image motion detection algorithms is object detection. While not specifically designed for this purpose object detection is capable of identifying objects in an image or video sequence. By identifying the objects location between different frames a translation can be inferred. Machine learning networks such as neural networks are trained on a large datasets of thousands of images to recognise specific objects in specific scenes. For example computer chips on a conveyor belt to instruct a pick and place robot. However, this implies that they are constrained to the objects they are trained on, meaning this is not a general technique. While novel and interesting its lack of general application means the author will not consider it for the remainder of this paper.

2.3 Related Literature

2.3.1 Resolution limits to object tracking with subpixel accuracy

D.Mas *et al.* [14] derived the definition for the minimum requirement for motion to be detected in a digital image as being the motion that causes 1 pixel to change state. The optimum target design to increase the probability of a pixel change for small displacements was investigated. The target designed in this paper involved dividing each pixel into subpixel arrays and sparsely populating these arrays with dots such that each excited pixel is surrounded by unexcited pixels.

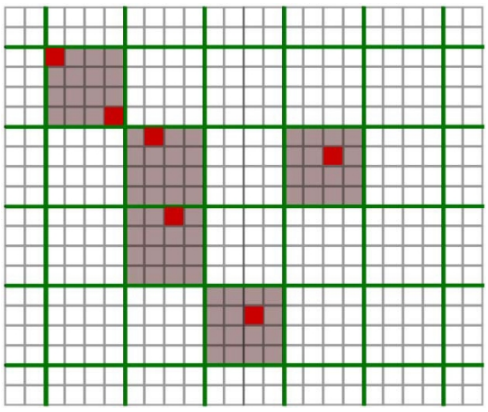


Figure 2.5: Target design using sparsely populated subpixel arrays [14].

The authors concluded that applying this method would theoretically increase the ability to track objects with subpixel accuracy by several orders of magnitude delivering micropixel accuracy, and was shown to be less dependent on the resolution of the observing camera.

2.3.2 Realistic limits for subpixel movement detection [15]

In this paper, set out to determine the limit to subpixel motion detection mathematically in a theoretical perfect situation. The authors acknowledged that practically this accuracy is not achievable due to limitations of real world imaging systems. Further to this the authors state that a shift of at least $1/2^B$ is required to change the state of 1 pixel, where B is the number of bits used to represent each pixel. This is proven below in the following equations. Using an expression for the squared flat energy profile to describe the normalised response, the following equation was derived.

$$E \times p \times \delta_x \geq 1/2^B \tag{2.8}$$

Where E is the energy of the squared profile, p is the side length of the pixel sensor, δ_x is the shift in the x direction. The expression for a pixel shift is determined as follows.

$$\delta_x \geq \frac{p}{2^B}, \quad \frac{\delta_x}{p} = \delta_p \geq 1/2^B \tag{2.9}$$

The authors concluded from their results that subpixel motion can be detected up to sub pixel size equal to the inverse of the number of quantisation levels in the image. Hence the limiting factor for motion estimation will be the camera not the numerical algorithm.

For example, for an 8 bit imaging system, the theoretical minimum translation that can be detected is $\delta_p = 1/2^8 = 1/256 \approx 0.004$ pixel.

2.3.3 A robust edge detection method with subpixel accuracy

This paper by Q.Sun et al [16] proposed using smoothing spline algorithm of Reinsch which minimises the total square curvature of the spline to define the edge model.

$$\int [f''(x)]^2 dx, \text{ under the constraint } \sum_{i=0}^{n} \left[\frac{f(x_i) - y_i}{\delta y_i} \right]^2 \leq S, \quad f \in C^2[x_0, x_n] \tag{2.10}$$

where $f(x)$ is a smoothing spline function; y_i is the input series; δy_i is a series of weights, and S is a scaling parameter. The quantities of δy_i control the extent of smoothing and are implicitly rescaled by varying S. The results from this new model are then fed into the standard one dimensional gray moment formula whereby a edge line equation can be derived.

To determine the stability and robustness of this new approach this algorithm was ran on the same 11 x 11 pixel image, with increasing levels of white Gauss noise and speckle noise which is introduced by environmental noise and imaging noise respectively, in the real world. The measure variance of the improved method is $2.1918e^{-06}$ while the variances of conventional gray moment and the space-moment methods are $7.3087e^{-06}$

and $1.0860e^{-04}$ respectively. It can be seen that the improved method is more stable and is more robust against noise in the image. In the experiment, a computer with a Pentium-2.3 GHz CPU was used to run the algorithms implemented in Matlab 2012. By testing the same 11 $\times$ 11 pixel image, the experiment showed a run time is 0.004 s for improved gray-moment operator, while the gray-moment and spatial-moment operators' took 0.0024 s and 0.0011 s respectively.

2.3.4 Image Based Subpixel Techniques for Movement and Vibration Tracking

D. Mas *et al.* [17] demonstrated a method for movement and vibration tracking through image based subpixel techniques which consists of multi-level thresholding in order to track movement in the observed object. The decision to use elliptical targets was made such that, once the target was detected, its contour can be fitted to the geometrical figure. Its centre can then be calculated from the obtained parameters for both the original and translated images.

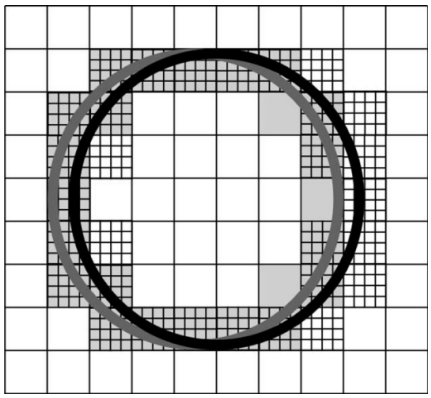


Figure 2.6: Elliptical target for sub pixel tracking[17].

The authors applied other methods including centroid tracking and sparsely populated subpixel arrays in order to track the observed object. These systems use subpixel resolution to enhances the spatial resolution. The authors concluded that this allows their implementation with cheaper lower resolution cameras.

Chapter 3

Proposed Method

The proposed method uses the fact that image sensors average the light falling over each pixel and return this value. For example with a dark object moving a subpixel translation in the positive x direction over the boundary between pixels, the pixels to the left of the object will get gradually brighter while the pixels to the right of the object will get gradually darker. By measuring the change in these areas between images we can estimate the translation that has occurred between them. Each image has a unique relationship between translation and the respective row or column total. This can be measured in simulation to calibrate the method and is related to the composition of the image, namely the number of edges and the difference from one side of the edge to another.

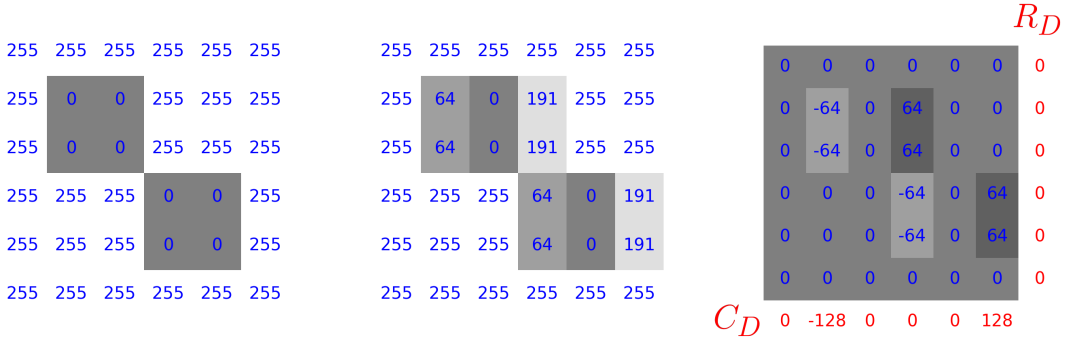
The proposed method begins with the inputs which are to be compared. These inputs are used to compute the difference matrix, D, as shown in the following equation $D = I - I_{\delta_x}$ where D denotes the difference matrix between I, the original image, and I_{δ_x} the translated image. This difference matrix is then used to compute the sum of the values in each row, R_D, and column, C_D, of the difference matrix as shown in the following equations:

$$R_D = \sum_{j=1}^{N} D_{ij}, \quad C_D = \sum_{i=1}^{N} D_{ij} \tag{3.1}$$

Following the computation of the summed rows and columns, the total magnitude of the area of the summed rows, T_{R_D}, and columns, T_{C_D} , is computed as shown in the following equations,

$$T_{R_D} = \sum |R_D|, \quad T_{C_D} = \sum |C_D| \tag{3.2}$$

T_{R_D} acts as an indicator of motion in the y axis and T_{C_D} shall act as an indicator of motion in the x axis. For motion in 1 axis, one should be significantly larger than the other and for equal motion in both axes they should be the same.



(a) Original Image. (b) Translated Image. (c) Difference Matrix.

Figure 3.1: Estimating Translation after a 0.25 pixel x translation. Pixel values overlaid indicate the brightness of each pixel using an 8 bit value ($2^8 = 255$ values) where 0 indicates a pixel with no light falling on it (black) and 255 indicates a saturated pixel (white).

For this image we can compute T_{R_D} and T_{C_D} to be 256 and 0 respectively. This indicates that there has been motion in the x direction and none in the y direction as we expect. This image has a t_x/T_{R_D} value of 1024 and hence $256/1024$ gives us our translation of 0.25pixel.

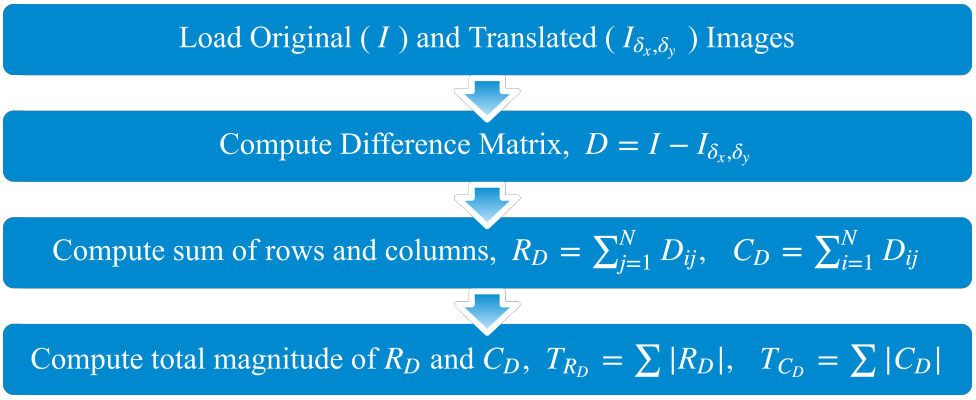
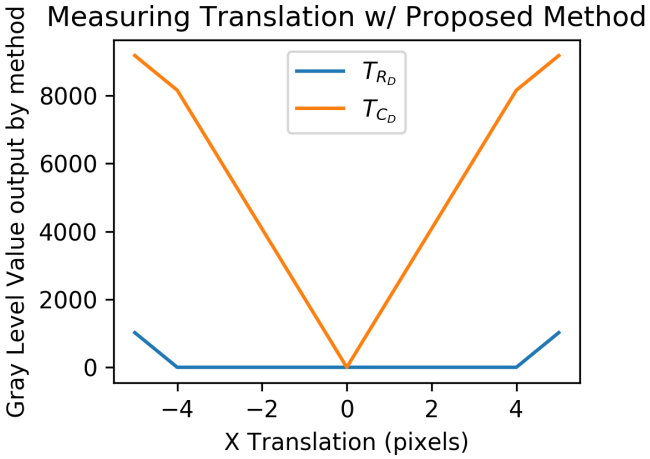


Figure 3.2: Proposed Method Flowchart.

3.1 Operation

Below is a graph of the output of the proposed method versus translation in the x direction tested on the a simple 6×6 image. Note the constant slope from -4 to 4 pixel translation including sub pixel motion detection. With a larger then 4 pixel translation in either direction the slope changes and the method falsely reports translation in the y direction. This indicates that the proposed method only has a reliable estimation of the motion within a ± 4 interval for this image. This maximum translation property will be investigated in section 7.2 and a method of detecting when the methods output can no longer be relied up will be demonstrated in section 5.3.

(a) Test Image



(b) Proposed method versus actual translation.

Figure 3.3: Translation estimation of the proposed method

The relationship between the image translation and the proposed method output can be easily seen in the slope of the graph. This image has a ${}^{t_x}/_{T_{R_D}}$ value of 1024 for example. This value can be derived by digitally translating the image and noting the slope or a target with a known ${}^{t_x}/_{T_{R_D}}$ can be attached to the moving object.

3.2 Previous Thesis

This project was taken on in 2018 by Christopher Duignan in partial fulfilment of the requirements for a masters degree of Electronic & Computer Engineering[18]. His results serve as the basis of this years project and are valuable to investigate.

3.2.1 Comparison - Gaussian Image

Correlation Using a quantised Gaussian function to represent an image as the test both the correlation algorithm and the proposed method were tested to over a range of 1000 translations, from 1×10^{-6} to 1×10^{2} pixels in the x direction, logarithmically spaced.

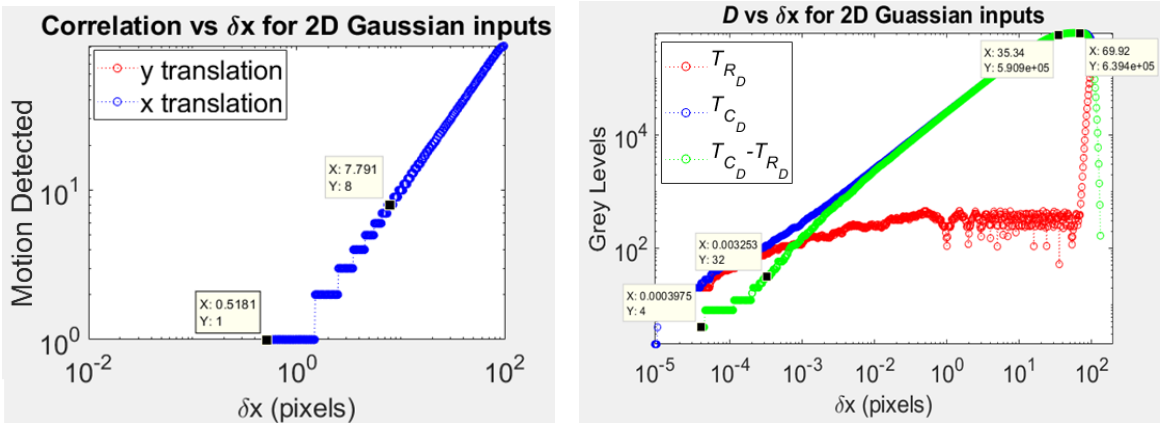
Duignan noted in their testing of the correlation method that the minimum translation detectable was 0.5pixel but that after this threshold the relationship between detected motion and translation was linear with steps of 1pixel, indicating the minimum resolution for the method was 1pixel. Also the correlation method does not falsely detect movement in the y direction.

Proposed Method

T_{C_D}represents the motion in the x axis and T_{R_D} represents the motion in the y axis. As the translation is in the x axis, T_{R_D} is treated as noise. The only source of noise in this simulation is that of quantisation and this noise is subtracted from T_{C_D} to obtain the true detected x motion.

Duignan noted in their testing of the proposed method that the proposed method has subpixel resolution and at 4×10^{-5} the motion detected overcomes quantisation noise. Duignan noted that before this point it is not possible to determine motion from noise. And that the relationship between motion estimation and translation becomes linear at 3×10^{-4}. The maximum translation detectable was determined to be 3.5 standard deviations (35 pixels) when the overlap between the Gaussians becomes marginal. At

this point the lack of overlap between the Gaussians causes the entire magnitude of the Gaussian to be included in the summing of the rows which leads to the difference between T_{C_D} and T_{R_D} to be null.



(a) Correlation method. (b) Proposed method.

Figure 3.4: Motion estimation methods applied to a Gaussian function.

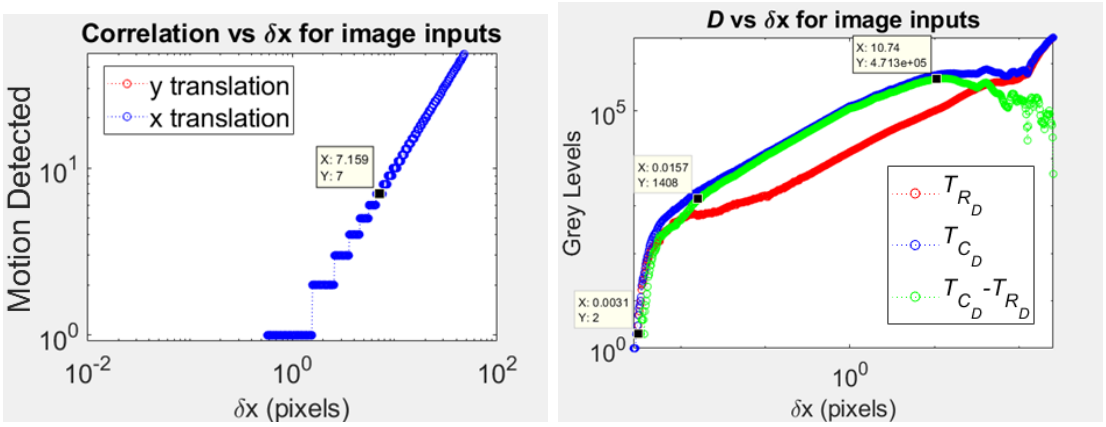
3.2.2 Comparison - Digital Image

Applying the correlation method to Digital Images was noted to not effect the accuracy of its output.

The author noted that proposed method suffered a marked deterioration in the proposed method's performance in the sub pixel region with the minimum detectable translation being 3×10^{-4} pixel in the x direction and a reduction of the maximum detectable translation. With the linear range beginning at 1×10^{-3} and the maximum detectable translation being 1×10^{1} pixel.

3.2.3 Robustness to Noise

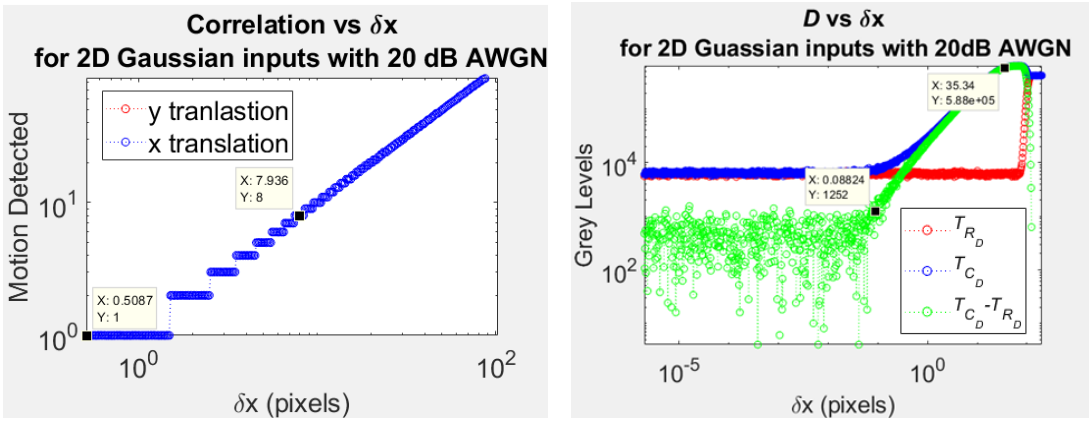
Following the introduction of noise, much of the region of sub pixel motion detection is lost in the proposed method, whilst the correlation method remains largely unaffected. The proposed method does maintain a distinct advantage in the sub pixel region, detecting motion up to 100 times smaller than the correlation method.



(a) Correlation method. (b) Proposed method.

Figure 3.5: Comparison of motion estimation methods applied to a digital image.

Despite the proposed method being sensitive to the addition of noise, the correlation remained practically unaffected which indicates that it is very robust with respect to noise. The range of the proposed method was halved in the case of digital image outputs whereas the correlation method performed almost identically to before the addition of the noise.



(a) Correlation method. (b) Proposed method.

Figure 3.6: Comparison of the effect of 20dB of noise on the motion estimation methods.

3.2.4 Comparison - Computational Expense

The author noted there is a substantial difference in overall simulation time between the 2 methods stating that the proposed method was 2 orders of magnitude faster than the correlation method.

Input	Proposed method	Correlation method
Random Matrix [150x150]	0.103s	15.398
2D Gaussian [603x603]	3.162s	35m 42.23s
Digital Image [768x768]	7.099s	10h 14m 11.27s

Table 3.1: Compute time of motion estimation methods vs input image size

3.2.5 Experimental Testing

The proposed method was experimentally proven to be capable of estimating motion in in 2 axes simultaneously.

The author also investigated several physical parameters of a real world imaging system and their effect on the output of the proposed method. The parameters tested were

- 2D axial translation
- Target focus
- Target design
- Target Illumination wavelength,
- Background lighting intensity
- Image capture time interval
- Sample size (images captured)

It was also concluded that the proposed method is robust to variation of many of the factors which are encountered in a digital imaging system. The only factor found to have a negative impact on the effectiveness of the proposed method was using a random target, whereby the proposed method output began to saturate for translations greater than a pixel.

Chapter 4

Implementation

The proposed method was implemented in Python for ease of testing and to allow rapid development. It being completely open source also means that there is no barrier to entry for other researchers looking to use the code developed with thesis. NOTE: for the sake of being succinct all sanitisation, error detection & handling and debug code has been removed to provide a minimum working example for the reader to follow. All code for this thesis can be found on the authors GitHub repository linked in appendix C

The proposed method is implemented with the following code:

```
def ProposedMethod(Original, Translated, ErrorCheck):

    #Compute Difference Matrix
    SubIm = Original - Translated

    #Calculate Row and Column totals
    RowSumVals = [sum(row) for row in SubIm]
    ColSumVals = [sum(col) for col in SubIm.T]

    #Compute absolute sum of row & col sums
    RowTot = sum(map(abs,RowSumVals))
    ColTot = sum(map(abs,ColSumVals))

    #Calculate difference between positive and negative areas
    ErrorCheck = np.average(SubIm)

    return RowTot, ColTot, ErrorCheck
```

Figure 4.1: Python implementation of the proposed method

Originally the entire process was implemented using the Python Image Lib Library however it is easier to work with the image as a matrix and therefore the project was rewritten to use the Numerical Python (NumPy) library. Both versions can be found on the aforementioned GitHub repository. One of the difficulties encountered is that while images are stored as commonly stored as 8 bit unsigned integer matrices (0-255) when we calculate the difference matrix we will contain both positive and negative values [1]. Hence why 16 bit signed integer matrices are used to represent the images and the difference matrix. This could be further optimised as technically for an 8 bit value the maximum possible range is [-255, 255] which can be stored in 9 bits. This was not implemented because creating and using custom data types would require tweaking nearly every available library function otherwise available to us in NumPy as well as making storing the values very difficult as 9 is not a power of 2. This method can be extrapolated to work with any number of bits but for the sake of being straight forward this thesis will focus mainly on 8 bit images.

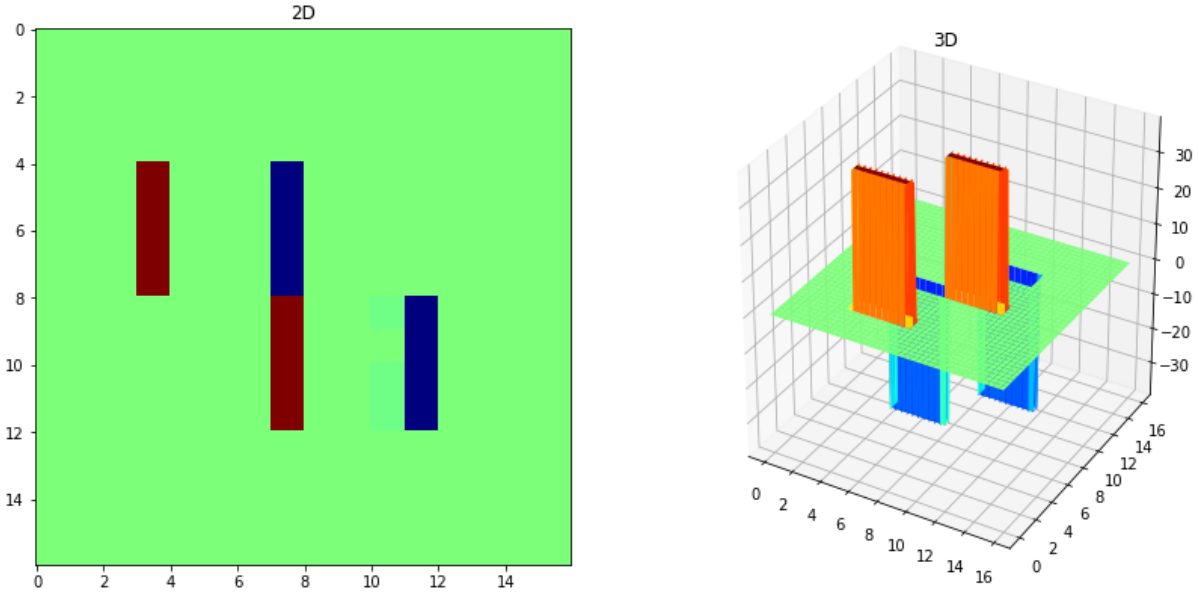


Figure 4.2: 2D and 3D visualisations of a difference matrix

[1] There are ways around this such as offsetting the values by half the maximum value and scaling the image 2x and then converting back after. However the resolution is halved because of this and noise is introduced in the rounding to nearest whole integer

4.1 Preprocessing

Before we can work with the Image it must first be imported and converted to a format we can work with. Common formats for images are the Joint Photographic Experts Group (JPEG) and Portable Network Graphics (PNG) formats. To work with these formats the Python Imaging Library (PIL) was employed.

First each pixel is converted to a single brightness value (greyscale) as discussed in. To account for the fact that our eyes are more sensitive to particular wavelengths of light we will employ the ITU-R 601-2 luma transform. This is defined as :$L = R * 299/1000 + G * 587/1000 + B * 114/1000$.

While in the real world PNG images will not contain any transparent pixel values often our test images will be provided in a vector form which when converted to PNG will result in transparency values. To account for this when we detect a ".png" file extension we will check if the image is in the form RGBA, where the fourth added channel is transparency. If this is true transparent pixels are overwritten with white values.

```
from PIL import Image

def importImage(Image_filepath):
    image = PIL.Image.open(Image_filepath)

#explicitly set transparent pixels to white.
    if Image_filepath[-4:] == ".png": #if the fileextension indicates it is a png
        pixel_data = image.load()

        if image.mode == "RGBA":
          # If the image has an alpha channel, convert it to white
          for y in range(image.size[1]): # For each row ...
            for x in range(image.size[0]): # Iterate through each column ...
              # Check if it's opaque
              if pixel_data[x, y][3] < 255:
                # Replace the pixel data with the colour white
                pixel_data[x, y] = (255, 255, 255, 255)
```

Figure 4.3: Python implementation of image preprocessing.

4.2 Increasing throughput

As we are trying to provide the maximum throughput per second there are several avenues we can explore. The brute force option is to utilise more powerful and expensive hardware, however, often this is not possible due to budgetary or power requirements of different projects. Hence, the algorithm must be intelligently written to optimise it.

There are two common avenues to explore when trying to increase the throughput of a program. Parallelising the algorithm to use multiple processors, or graphics processing units (GPU) or use a lower level language with less overhead.

4.2.1 Using a more optimised programming language

The first option is to implement the algorithm in a language with less overhead. Writing the program in C or Assembly programming language would provide significant increases in throughput due to the much more efficient optimisations possible. This is mainly due to the fact that C and Assembly is compiled and statically typed. These mean that the code can be optimised before it is ran and the optimisation algorithm knows what format each variable is stored as [2]. As this simply means reimplementing the algorithm with a different programming language it provides no real knowledge gain and therefore was not pursued.

4.2.2 Multiprocessing

Implementing multiprocessing for this algorithm can be carried out with multiple processors in mind or GPU units. It was decided not to focus pursue GPU optimisation [3] as low power systems often do not have discrete GPUs but will often have multiple cores. As a note however

[2]Technically the NumPy package runs compiled C code in the back-end and uses declared static variables (eg the 16bit integer data types) so a lot of this overhead has already been removed.

[3]Optimising the code for execution on a GPU would involve writing the algorithm to specifically use matrix operations. This work has already been done and the NumPy arrays used are in fact matrices and much of the code is already written to use the inbuilt matrix operations NumPy provides.

Implementing a multiprocessing capable version of the proposed method requires several tweaks to the operation of the algorithm but fundamentally can be understood as evenly splitting the image into sub images into N rows[1] (where N is the number of processors). Processing these individually before collating the results back together. For each sub image the difference matrix can be created and the column and row sum values (R_{D_P}, C_{D_P}) can be calculated. As we have entire rows we can calculate the row total for the sub image. The average of the sub image difference matrix can also be calculated. All these sub images can be processed in parallel which represents the bulk of the computing. Yielding a theoretical speed-up of N available for this step. There is more overhead associated with this however such as forking the processes and therefore there will not be a perfect N fold increase in throughput. Once these values are calculated for the individual sub images the values for the total image can be simply calculated in a comparatively small number of operations.

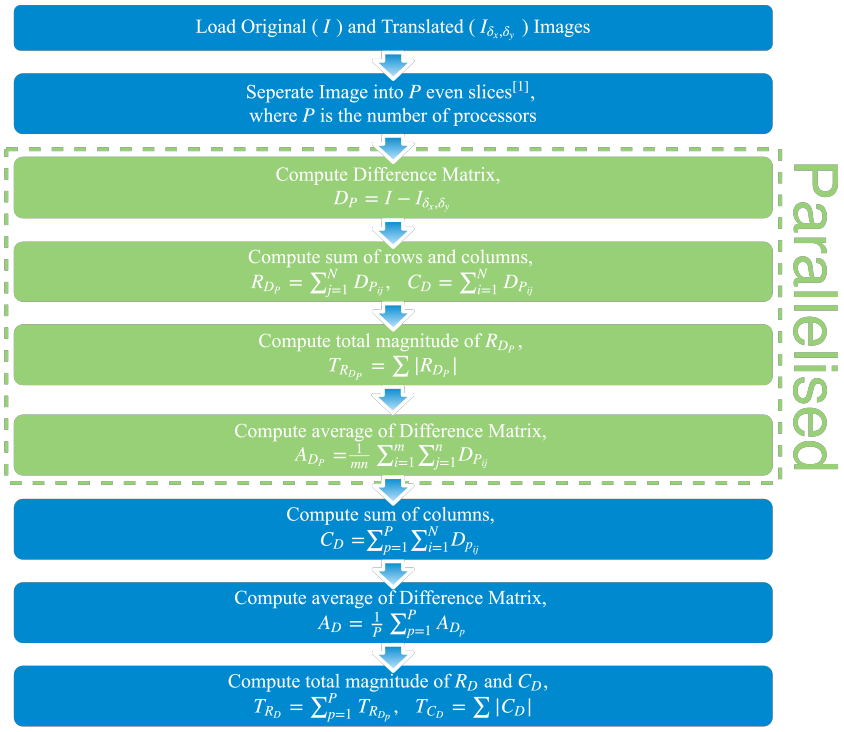


Figure 4.4: Flowchart of a multiprocess capable version of the proposed method.

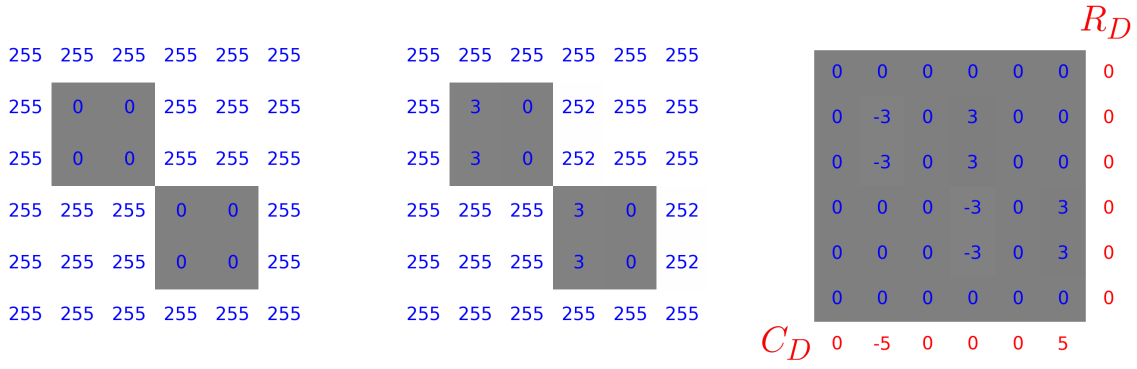
[1]If the image dimension is not evenly divided the value is rounded down to the nearest integer and the remaining rows are put into the final sub image

Chapter 5

Method Advantages

5.1 Spatial Resolution

As defined by Mas *et al,*[14] the minimum detectable digital image translation is that whereby a single pixel changes value by the minimum amount. This is determined by the number of bits used to represent each pixel. For example, for a 8bit image there are 256 values to choose from. The proposed method can detect this smallest change and hence has a minimum detectable translation of $1/2^b$. ie $1/256$ of a pixel for an 8 bit image.



(a) Original Image (b) Translated Image (c) Difference Matrix

Figure 5.1: A $1/100$ pixel translation is detectable with this method, $T_{C_D} = 10$

.

5.2 Robustness to Noise

As noise is normally a random process with a mean of zero (Gaussian, Poisson) an additive or subtractive process is able average the noise out across an image. Meanwhile a multiplicative process (like convolution) will exacerbate the effect of noise. With increasing image size the proposed method is more resistant to noise meanwhile the convolution method becomes more sensitive to noise.

5.3 Error Detection

As an image moves in a direction (for example positive x) the pixels behind it change and the ones in front of it change by an inversely proportional amount. Hence these two areas in the difference image cancel out. Deviation from zero of this averaged value can be used to quantify the effect of noise and edge effects on the image and quantify the amount of error in a motion measurement.

$$\text{Error Check Value} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij} \tag{5.1}$$

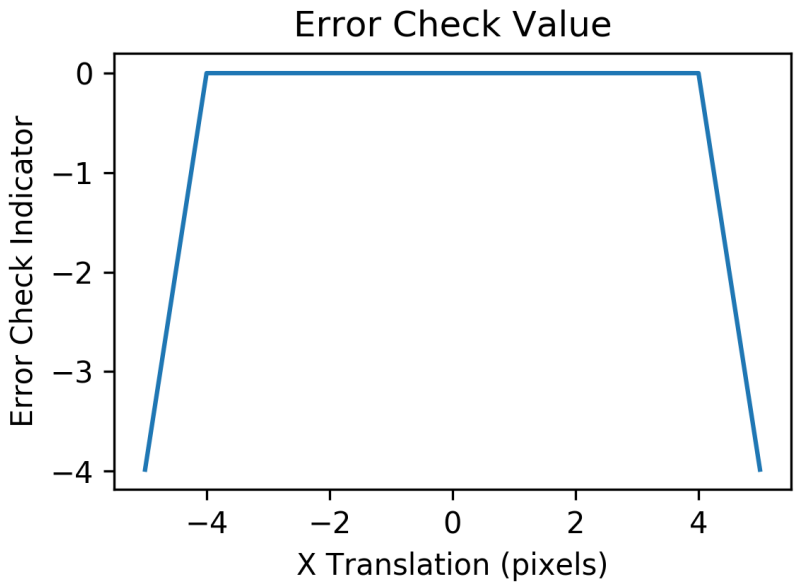


Figure 5.2: Error check output indicating an error in the motion estimation in figure3.3b when there is a larger then 4 pixel translation in either direction.

Chapter 6

Computational Expense

To determine the efficiency of an algorithm, we must consider how difficult it is to compute versus with different parameter settings, eg bit depth image size. This allows us to relatively compare how two algorithms will perform in real world applications independent of the hardware used. In computer science, big O notation (Θ) is used to classify algorithms according to how their running time or space requirements grow as the input size grows.

6.1 Compute

A way of evaluating how long an algorithm will take to compute is to evaluate how many arithmetic operations are required for a general case. For our initial investigation we will assume application data loading and other operations have negligible effect on the computation time due to the relative small size of individual images and the speed of modern storage technologies (RAM, SSD's etc).

To simplify this investigation and the notation required we will consider a square image of dimension $N \times N$ pixels where the image is greyscale.

6.1.1 Proposed Method

Computing the difference matrix entails subtracting the pixel intensity value of the translated image from the original image at every pixel location. Hence requiring $N \times N$ subtractions. Which we can equate to additions to simplify our notation. Summing the rows will require N additions per row for N rows, $N \times N$ and likewise for the columns. Calculating the sum totals for the rows requires N more additions and likewise for the columns.

Hence we have $N^2 + 2N^2 + 2N = 3N^2 + 2N$ addition operations

6.1.2 Cross Correlation

Before we test the image with this method we must first zero-pad the original image with a border equal to edge length of the window. Zero Pad entails adding pixels with a value of 0. This reduces aliasing (the sharp change of intensity at the edge of the image showing up as high frequency peaks when using frequency domain methods) and allows to test the edge positions where the window will pass outside the bounds of the images.

A single cross correlation requires each pixel of the window to be compared to the image. Thus requiring $W \times W$ multiplications.

This window must be tested at every position within the image of which there are $N \times N$. Hence, the direct method requires $(N \times N) \times (W \times W) = N^2W^2$ multiplications.

Frequency Domain Improvement

As this is effectively a convolution we can utilise the properties of the Fourier space to reduce the computational expense.

Firstly the fast Fourier transform (FFT) of the image must be computed. This is done by applying an FFT to each column separately, and then to each row. In general we can assume that a FFT for N data points takes approximately $2N \times log_2(N)$ multiplications. As we have N rows and N columns we must perform $2N(2N \times log_2(N))$ multiplications.

Next the FFT'd image is multiplied by the Fourier transform kernel at every pixel position. Requiring $N \times N$ multiplications. However as we are multiplying complex values of the form $a+bi$ this entails $4N^2$ real multiplications following $(a+bi)^2 = a^2+b^2+ab+ba$.

Finally we must employ the inverse Fourier transform on the data to get back to sensible information. requiring another $2N(2N \times log_2(N))$ multiplications. Hence, we require $2(4N^2 \times log_2(N)) + 4N^2 = 4N^2(2 \times log_2(N) + 1)$ multiplication operations.

As this is a frequency domain method zero-padding in the context of correlation is done to ensure the process yields linear instead of circular convolution/correlation in the frequency domain. Therefore, preventing aliasing when we inverse Fourier transform back to the time domain. Hence, the images are zero padded in both directions and can simply be considered as starting with a $3N \times 3N$ image.

To match the resolution of $1/2^b$ pixels (where b is the bit depth of each pixel) of the proposed method the correlation function must super-sample the image 2^b times.

6.1.3 Comparing Methods

To allow more direct comparison we will set the window size using in the correlation to be equal to that of the image edge length ($W = N$). Obviously applying a smaller window size will result in a speed up of the correlation method but this would defeat the point of using the high resolution sensors in the first place.

In this case we define the images as $N \times N$ pixel b-bit greyscale.

Method	Number of Operations
Time Domain Correlation	N^4
Frequency Domain Correlation	$4(3N)^2(2 \times log_2(3N) + 1)$
Proposed Method	$3N^2 + 2N$

Table 6.1: Comparing each Algorithms Computational Expense

To compare the impact of multiplication vs addition we can refer to how many individual operations are required.

Addition	Two n-digit numbers N, N
Schoolbook addition with carry	$\Theta(N), \Theta(log(N))$
Subtraction	**Two n-digit numbers N, N**
Schoolbook subtraction with borrow	$\Theta(N), \Theta(log(N))$
Multiplication	**Two n-digit numbers**
Schoolbook long multiplication	$\Theta(N^2)$
Karatsuba algorithm	$\Theta(N^{1.585})$
3-way Toom–Cook multiplication	$\Theta(N^{1.465})$

Table 6.2: Comparison of standard arithmetic operations computational expense without specialised hardware

In table 6.2 we can see that additions can be equated to subtractions. This is easy to comprehend if we simply consider subtraction as firstly sign flipping the value to be subtracted and then continuing with a normal addition operation.

Method	Number of Operations
Time Domain Correlation	$\lceil b^{1.465} \rceil N^4$
Frequency Domain Correlation	$\lceil b^{1.465} \rceil 4(3N)^2(2 \times log_2(3N) + 1)$
Proposed Method	$(3N^2 + 2N)(log_2(b))$

Table 6.3: Comparing the computational expense of different motion estimation methods without specialised hardware.

Bit Depth	Proposed Method	Frequency Domain Correlation	Speed up
2	1.16e+03	1.67e+10	1.45e+07x
4	1.60e+03	2.38e+10	1.49e+07x
8	2.70e+03	4.20e+10	1.55e+07x
16	5.78e+03	9.48e+10	1.64e+07x

Table 6.4: Number of operations needed on a $100x100$px Image vs pixel bit depth without specialised hardware.

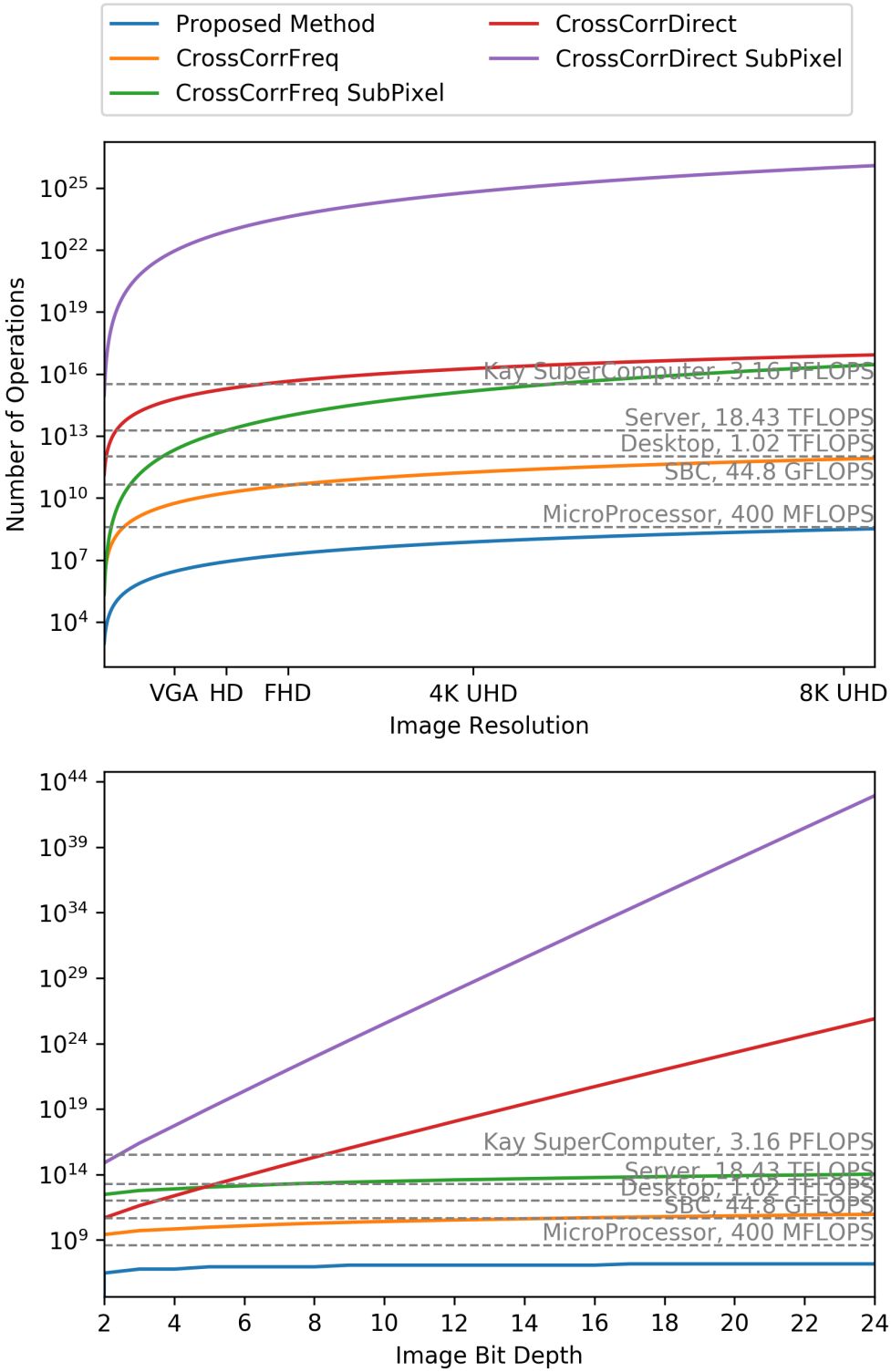


Figure 6.1: Comparing the computational expense of different motion estimation methods without specialised hardware. Pixel bit depth fixed to 8bits and Resolution fixed to HD (1280x720 pixels) respectively.

For example, using 1080p images the proposed method is theoretically 10^6 times faster then Frequency Domain Correlation. With modern hardware however there is nearly always specialised hardware to compute both multiplications and additions in one operation. Hence, we will can ignore the impact of multiplication operations over additions.

6.1.4 Sub Pixel Motion Estimation

Method	Number of Operations
Time Domain Correlation	$(2^b N)^4$
Frequency Domain Correlation	$4(2^b 3N)^2(2 \times log_2(2^b 3N) + 1)$
Proposed Method	$3N^2 + 2N$

Table 6.5: Comparing the computational expense of different motion estimation methods.

We can note that for supra pixel translations the computational expense versus the image size of the Direct Cross Correlation Method is approximately quartic and the Frequency Domain Cross Correlation & Proposed methods are approximately quadratic.

To get a grasp of how long the algorithms will take to compute on given hardware we can calculate the number of Floating Point Operations (FLOPS) the hardware can do per second. While the algorithms do not strictly employ floating point operations it is a good indication as floating point and integer operations can be treated much the same. To calculate peak theoretical performance of a system we first need to calculate peak theoretical performance of one node (server) in GFlops and than just multiply node performance on the number of nodes the system has. This can be described as: Node performance in GFlops = (CPU speed in GHz) × (number of CPU cores) × (CPU instruction per cycle (IOPS)) × (number of CPUs per node) [19].

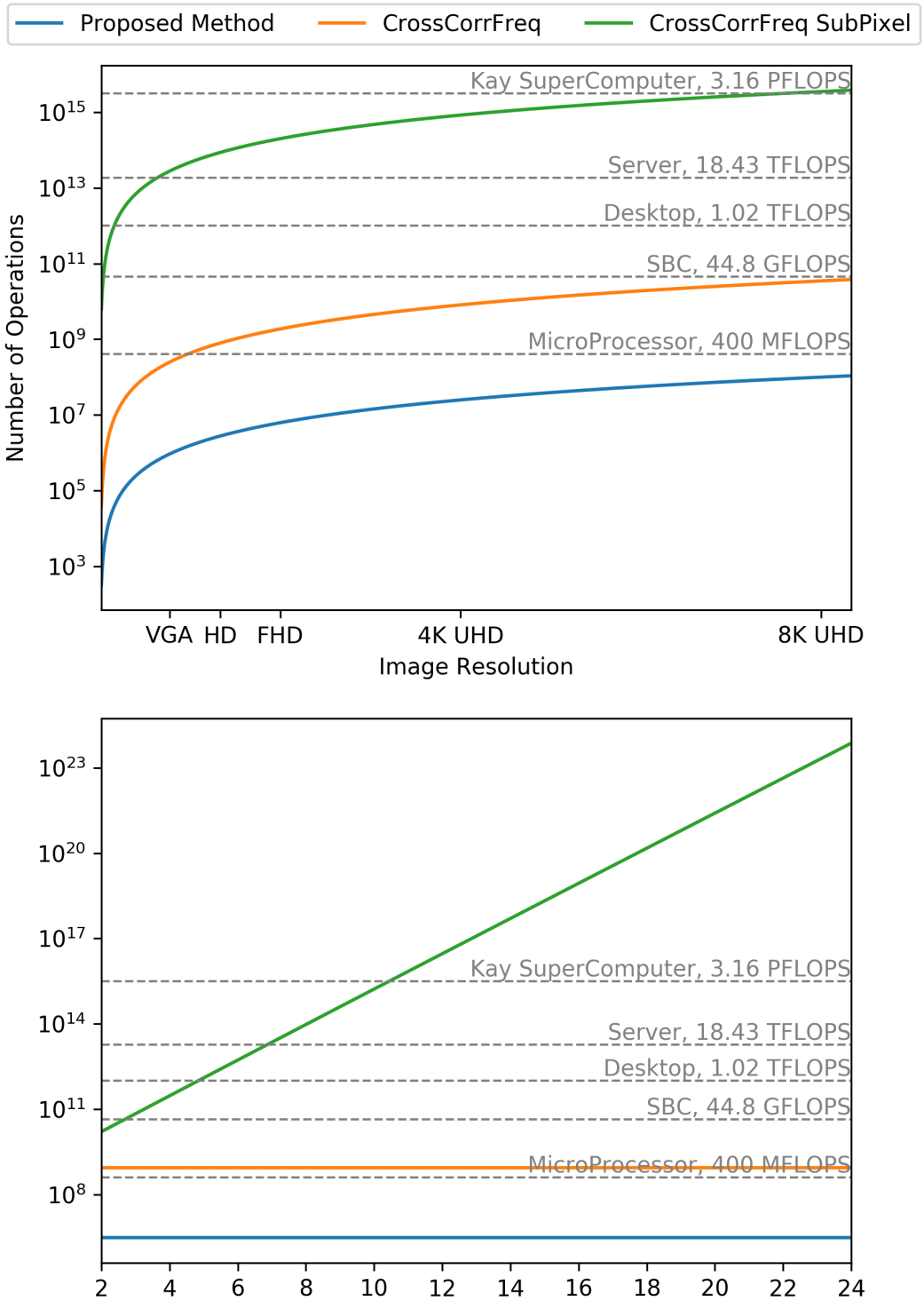


Figure 6.2: Comparing the computational expense of different motion estimation methods. Pixel bit depth fixed to 8bits and Resolution fixed to HD (1280x720 pixels) respectively.

Compute System	IOPS	CPU Cores	Clock Speed (GHz)	GFLOPS
Kay Supercomputer [20]	64	13440	3.7	3182000
Server [21]	64	96	3.0	18816
Desktop PC	32	8	4.0	1024
Single Board Computer (SBC) [22]	8	4	1.4	44.8
Microcontroller [23]	1	1	0.4	0.4

Table 6.6: Different System Hardware Specifications [24]

6.2 Execution Memory Requirements

Often there are significant accelerations to be had if an algorithm can be run from a faster storage device. With low power devices in particular having very restricted RAM memory. This can also often be a bottle neck for the algorithms execution if the data needed for the algorithm to run must constantly be swapped into and out of high speed memory as it is too large to be loaded at once. Hence, it is valuable to investigate the space requirements.

6.2.1 Proposed Method

In operation the proposed method keeps three arrays, the Original, Translated and Difference arrays in memory. It also stores row & column sums and total column & row sum variables in memory. This can be expressed as $3 \times ((N \times N) * b) + 2 \times (N * b) + 2 * b = b(3N(N+1)+2) \approx 3N^2b$ bits in memory. This is an theoretical best case value there will be overhead especially if the bit depth does not evenly fit into bytes.

6.2.2 Cross Correlation

The computation of the memory requirements for the cross correlation method are also very similar. For a subpixel resolution output there needs to be three arrays, the Original Translated and Similarity arrays in memory. To work with the convolution in the frequency domain the original and translated images both need to be zero-padded in both directions and then super-sampled up to match the subpixel resolution of the proposed method. This can be expressed as $3(3N2^b)^2$

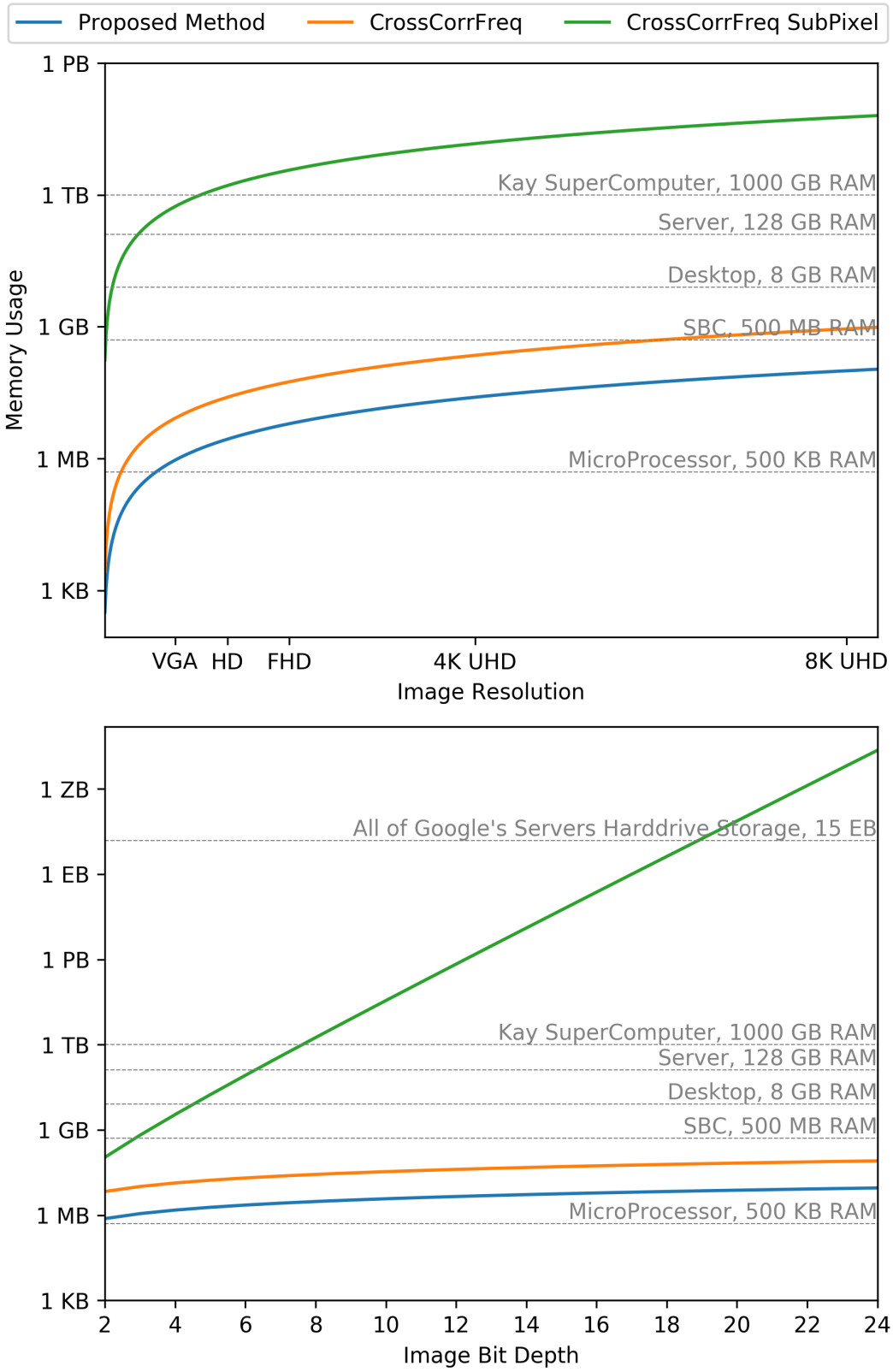


Figure 6.3: Comparison of different motion estimation algorithms storage expense. Pixel bit depth fixed to 8bits and Resolution fixed to HD (1280x720 pixels) respectively.

Image Size	Computation Duration (secs)	Frames per second
280 x 280	0.05	18.78
550 x 550	0.14	6.94
700 x 700	0.21	4.86
960 x 960	0.37	2.74
1200 x 1200	0.57	1.74
1440 x 1440	0.82	1.22
1500 x 1500	0.93	1.07
2880 x 2880	3.43	0.29

Table 6.7: Image size versus compute time and frames per second throughput for the proposed method, raw values

6.2.3 Real World Testing

Running each algorithm on hardware using the same images and recording the duration for them to run allows us to verify the relationship between image size and computational expense.

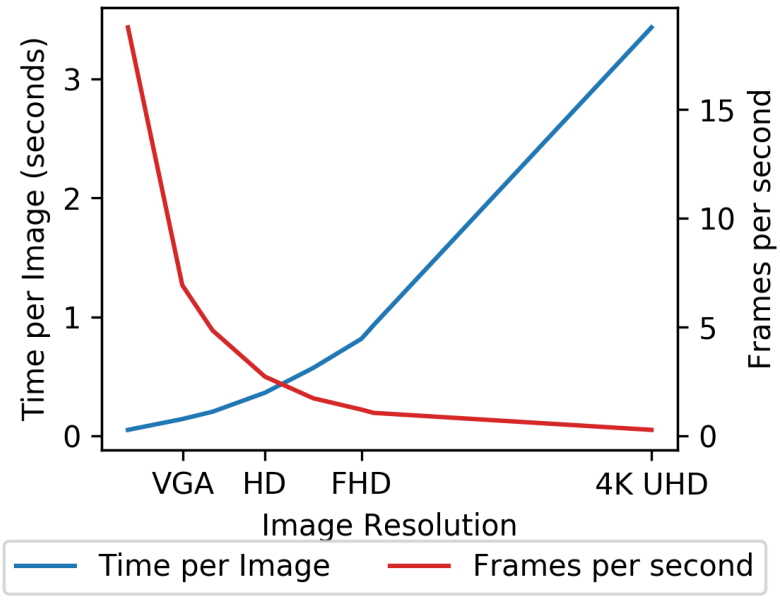


Figure 6.4: Image size versus compute time and frames per second throughput for the proposed method

Chapter 7

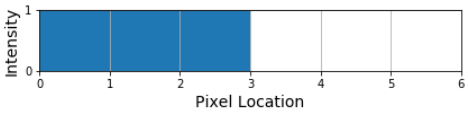
Method Limitations

To analyse the limitations of this method it is useful to consider a 1D case where we have each pixel take either a blank (0) or dark (1) value. We can use a simple step function to model an idealised edge.

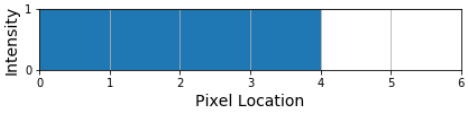
$$\text{Pixel Value} = \begin{Bmatrix} 1 & \forall x \leq 4 \\ 0 & otherwise \end{Bmatrix}$$

Then we can translate this function exactly 1 pixel in the positive x direction. Computing the difference matrix we can see the only difference is where the boundaries of the contiguous areas overlapped. With a single pixel difference we get a difference of 1px.

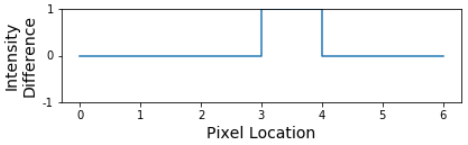
Hence we can see that the magnitude of the difference scales linearly with the translation and when we flip the translation direction the sign of the difference magnitude flips.



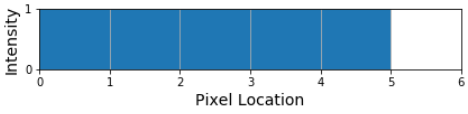
(a) Original Image.



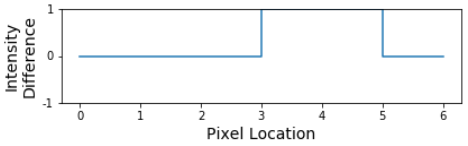
(b) Image Translated +1 Pixel.



(c) Image Difference, $\sum = 1$.



(d) Image Translated +2 Pixel.



(e) Image Difference, $\sum = 2$.

Figure 7.1: Simple Image Model

7.1 Translation Direction

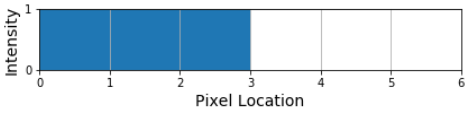
We can show that the sign of the difference is not dependent on the direction of translation but rather the composition of the image. This achieved by inverting the step function used to describe the edge.

$$\text{Pixel Value} = \begin{cases} 0 & \forall x \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

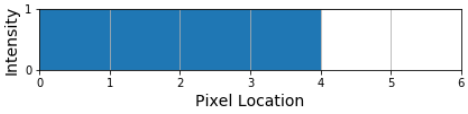
This is because the subtraction operation used to create the difference matrix is non commutative. Which means that a group of quantities connected by operators does not give the same result whatever the order of the quantities involved. Hence we can consider the Original and Translated Images as represented by matrices A and B respectively.

The original operation can be considered as $A - B$, however this equals the inverted operation $(-(B - A))$ e.x. instead of moving from the original to the translated image we are moving from the translated back to the original.

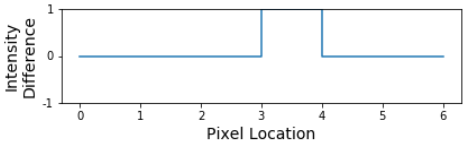
Hence, the proposed method fundamentally cannot determine which direction an object is moving only the magnitude of the translation.



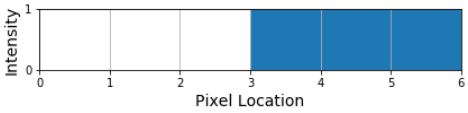
(a) Original Image.



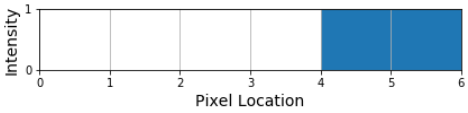
(b) Image Translated +1 Pixel.



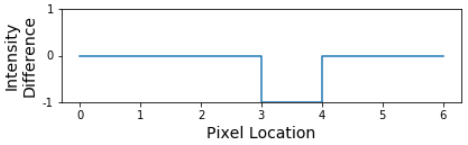
(c) Image Difference, $\sum = 1$.



(d) Inverted Image.



(e) Inverted Image Translated +1 Pixel.



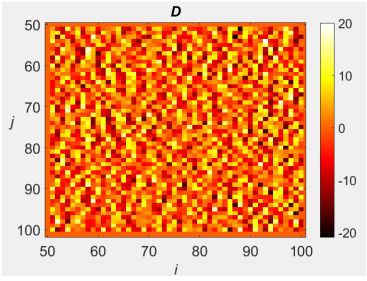
(f) Inverted Image Difference, $\sum = -1$.

Figure 7.2: Translation Direction Limitation.

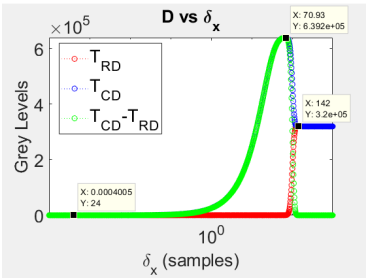
7.2 Maximum Translation

In previous analysis by Duignan [18] it was shown that after a certain amount of translation the method began to break down however this was not analysed to determine the root cause. The method was noted to perform well for subpixel movement, however, with larger translations it was heavily dependent on the image composition. Below two results from the thesis are presented to illustrate this point.

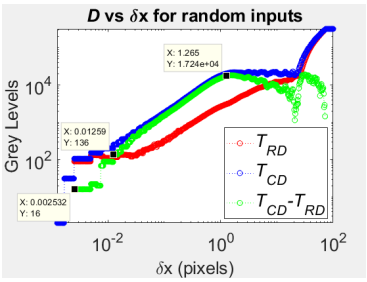
(a) 500 px^2 Coffee Bean Test Image



(b) Random Matrix Image



(c) Proposed Method Output



(d) Proposed Method Output

Figure 7.3: Motion Estimation Limitation [18]

With the Coffee Bean Image the method begins to breakdown at 71 pixels of translation which corresponds neatly to the average of x length of the coffee beans which are the continuous shapes which dominate the image. For the Random Matrix Image we can see the motion begins to break down at $\approx$ 1px of translation, this again corresponds to the dominant continuous size which is the pixels themselves. It appears that the dominant continuous area size is the determining factor for how large a translation the proposed method can measure.

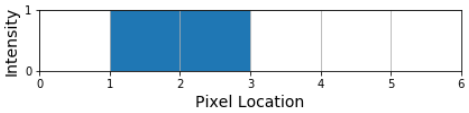
7.2.1 Cause of Motion Estimation Breakdown

We will use the step function,

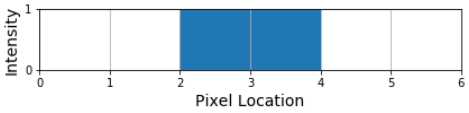
$$\text{Pixel Value} = \begin{Bmatrix} 0 & \forall & 1 \leq x \leq 4 \\ 1 & & \text{otherwise} \end{Bmatrix}$$

to represent a continuous block of 3 pixels, this is represents our original image (Figure 7.4a). Note that a 1 pixel translation causes the area od the absolute difference value to increase by 2. Figures 7.4b and 7.4d demonstrate the method working reliably. However once the translation becomes larger then the continuous block size, the area of the absolute difference value plateaus and if this were a repeating pattern the sum of the absolute difference value would begin to decrease again.

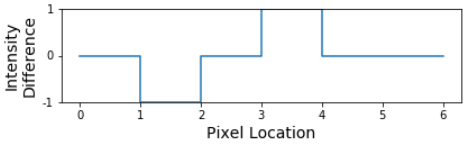
Hence, the proposed method is heavily reliant on the composition of the image above a translation of 1 pixel.



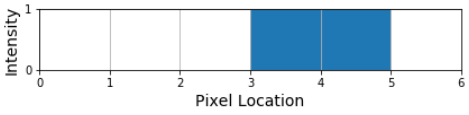
(a) Original Image.



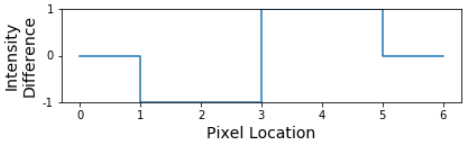
(b) Image Translated +1 Pixel.



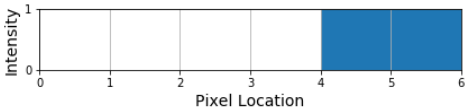
(c) Image Difference, $|\sum| = 2$.



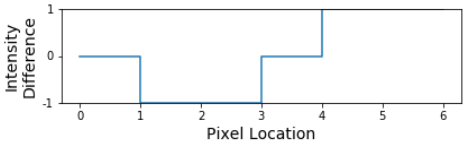
(d) Image Translated +2 Pixel.



(e) Image Difference, $|\sum| = 4$.



(f) Image Translated +3 Pixels.



(g) Image Difference, $|\sum| = 4$.

Figure 7.4: Motion Estimation failing.

Chapter 8

Conclusion

In this paper the fundamental operation of the proposed method was determined. The advantages it provides in terms of computational expense were calculated and compared to the correlation method and were shown to be massively reduced for the subpixel case. Fundamental limitations of the method were also derived and the root causes investigated. The method was shown to have a maximum translation for which it returns linear results. It was also shown that the proposed method is highly dependent on image composition for determining the direction of a translation. A method for detecting when the proposed methods output may not be accurate is also proposed.

Chapter 9

Further Work

9.1 Robustness to Noise

As the method is a additive method the noise with standard noise being a random process with a mean about zero, this process should be less susceptible to error as it is averaged out across the image. With increasing resolution noise should have less of an effect on the output. This idea needs further investigation and simulation to see if it holds and what factors effect its performance.

9.2 Combined Method

Combining the proposed method with a conventional correlation algorithm would allow much increased temporal and spatial resolution at much decreased computational expense. This method has the capability to track sub pixel movement for many images up until a threshold translation amount is reached. Then the proposed method can be verified with the correlation method and the original image reset and this cycle repeated again. Investigating the viability of this method and determining what factors influence the trade off is recommended.

9.3 Further investigation of the error checking method

The basic premise of the error check method has been outlined in this paper however adequate simulation to prove its effectiveness was not achieved before the culmination of this project.

Appendix A

Test Images Generation

In order to be able to test the various aspects of the proposed method we must have test images with which to work. To mitigate the effect of other factors in the process we will work with a completely digital process.

A.1 Generating Test Images

Working with subpixel translations immediately makes things complex as there are no common applications for this, therefore a new method for producing these test images had to be developed from scratch. In figure A.1 the method is outlined.

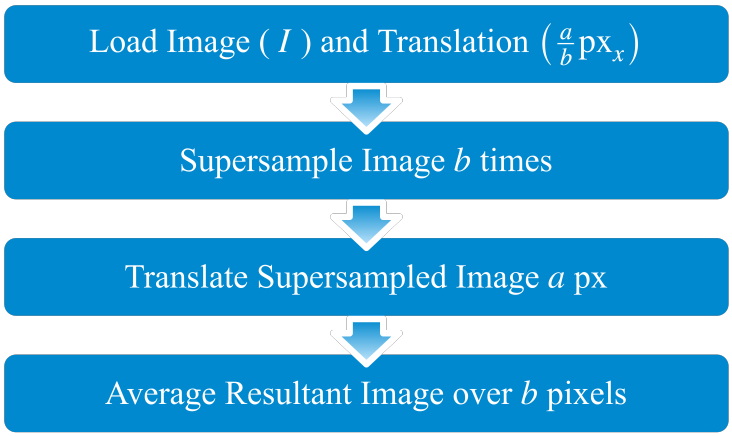


Figure A.1: SubPixel Translate Image Method Flowchart.

To enable us to properly account for edge effects (edge pixels being introduced and lost) we must first start with a larger image and window this. The size of the border needed is determined by the maximum translation we want. If we want to demonstrate a 10pixel translation we need at least a 10pixel border around the image.

Now to enable a subpixel translation we need to emulate how camera operates. To achieve this we supersample the image, shift the image a small value and then average the values over the pixel area to the required dimension.

For example, working with a 0.03pixel translation. This can be represented as $^3/_{100}$. Hence we will super-sample the image 100x, translate this 100x super-sampled image 3pixel and average back.

```
def subPixelTranslate(translation, Im, window_size):

    winx, winy = map(int,window_size)
    tx,ty = translation

    # Window around centre
    x_border = np.ceil(abs(tx))
    y_border = np.ceil(abs(ty))
    interWin_size= list(map(int,[winy+2*np.ceil(abs(y_border)),
    winx+2*np.ceil(abs(x_border))]))

    Im = window_on_centre(Im, interWin_size)

    #SuperSample
    px_tx, SupScale_x = floatToFrac(tx)
    px_ty, SupScale_y = floatToFrac(ty)

    Im = (Im.repeat(SupScale_x, axis=1)).repeat(SupScale_y, axis=0)

    x1 = round(SupScale_x*(np.ceil(abs(tx)))-px_tx)
    x2 = round(SupScale_x*(np.ceil(abs(tx))+winx)-px_tx)
    y1 = round(SupScale_y*(np.ceil(abs(ty)))+px_ty)
    y2 = round(SupScale_y*(np.ceil(abs(ty))+winy)+px_ty)

    #WINDOWING
    Im = Im[int(y1):int(y2), int(x1):int(x2)]

    #AVERAGING OVER AREA
    Im = averageSubArrayArea(Im, (SupScale_x, SupScale_y), DEBUG)

    return Im
```

Figure A.2: SubPixel Translate Image Code Snippet

Appendix B

Using Cloud Compute

In order to work with image processing and test new ideas often times you need to experiment with different things to try find what works and doesn't. When running a single experiment can take upwards of an entire day to run on a modern laptop a new approach is needed. This is where Cloud Compute comes in, allowing us to briefly use very very high power machines to run our experiments and get results back. For instance much of the Image generation was ran on high power Google Compute Instances.

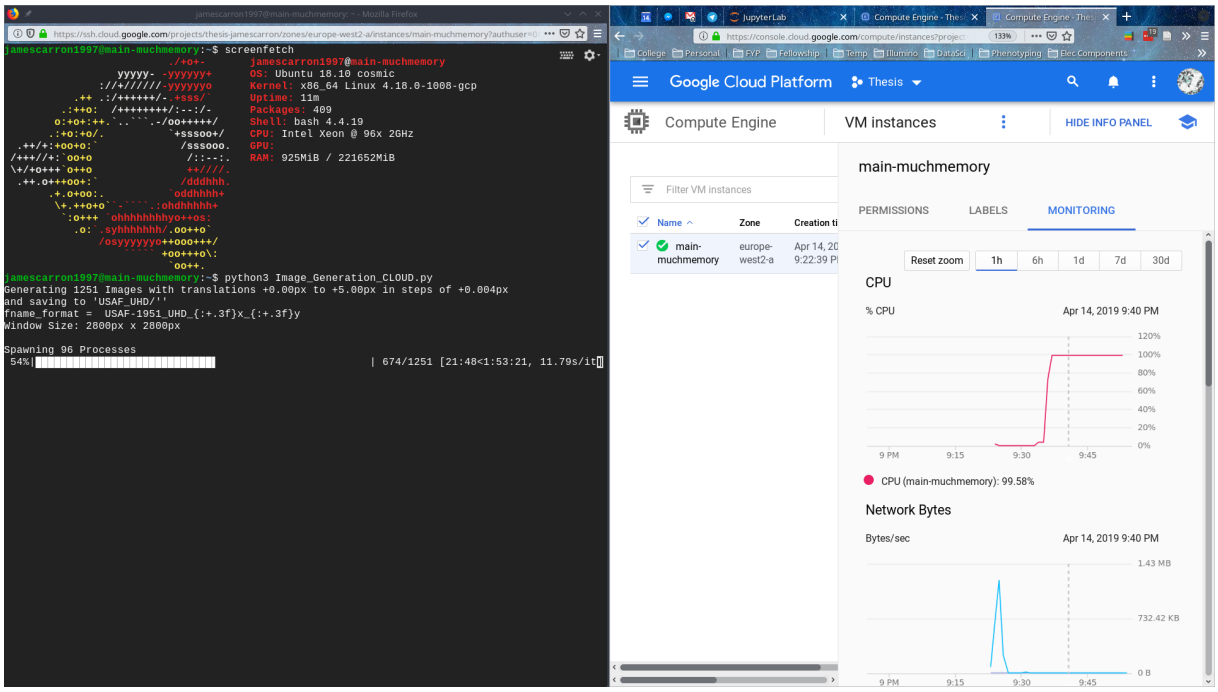


Figure B.1: Running the image generation code on a 96 core 200Gb RAM cloud instance

Appendix C

Code Snippets

Below are high level snippets of the code developed over the course of this project. NOTE: for the sake of being succinct all sanitisation, error detection & handling and debug code has been removed to provide a minimum working example for the reader to follow.

The full code for this project will be available on GitHub here:
https://github.com/JamesCarron/SubPixelMotionEstimation.

Figure C.1: QR code linking to the GitHub Repository for this project

```
def ProposedMethod_MULTITHREADED(Im1, Im2, NUM_THREADS=2):

    div_val = int(Im1.shape[0]/NUM_THREADS)

    #generate a list of the subarrays to pass to the multiprocessing pool
    #Ims = ((Im1_Slice1, Im2_Slice1), (Im1_Slice2, Im2_Slice2), etc)
    Ims = [(Im1[i*div_val:(i+1)*div_val][:], Im2[i*div_val:(i+1)*div_val][:]) for i in range
    (NUM_THREADS-1)]
    Ims.append( (Im1[(NUM_THREADS-1)*div_val:][:], Im2[(NUM_THREADS-1)*div_val:][:]) )
    #final one picks up any straggler rows if not even division

    pool = multiprocessing.Pool(processes = NUM_THREADS)
    func = functools.partial(ProposedMethod_PROCESS, ErrorCheck = ErrorCheck, DEBUG = DEBUG)
    data = pool.imap_unordered(func, Ims)
    pool.close()

    #Unpack info returned by the ProposedMethod_PROCESS function
    ColSumVals, RowTot_sub, ErrorCheckVals = zip(*list(data))

    #zip transposes list to add up the columns
    ColSumVals = [sum(col) for col in zip(*ColSumVals)]

    #compute absolute sum of row & col sums
    RowTot = sum(map(abs,RowTot_sub))
    ColTot = sum(map(abs,ColSumVals))

    ErrorCheckVals = np.average(ErrorCheckVals)

    return (RowTot, ColTot, ErrorCheckVals)
```

Figure C.2: Proposed Method Multiprocessed.

```
def ProposedMethod_PROCESS(Ims, ErrorCheck = False, DEBUG = False): #slave process

    #Unpack the images
    Im1, Im2 = Ims

    SubIm = Im1 - Im2

    RowSumVals = [sum(row) for row in SubIm]
    ColSumVals = [sum(col) for col in SubIm.T]

    #Calculate the sum of the rows
    RowTot = sum(map(abs,RowSumVals))

    ErrorCheckVals = np.average(SubIm)

    return (ColSumVals, RowTot, ErrorCheckVals)
```

Figure C.3: Proposed Method Multiprocessed Sub Process.

```
def window_on_centre(Im, window_size, SILENT = True):
    #determine the border length for each dimension
    border_len = [int((im_s-win_s)/2) for im_s,win_s in zip(Im.shape,window_size)]

    return Im[border_len[0]:border_len[0]+window_size[0],
    border_len[1]:border_len[1]+window_size[1]]

def averageSubArrayArea(Im, Area, DEBUG = False):

    j_max, i_max  = map(int,np.array(Im.shape)/(Area_y, Area_x))

    Averaged_Array = np.empty((j_max, i_max), dtype=np.uint8) #initialise array to shape

    for i,j in itertools.product(range(i_max), range(j_max)): #iterate through subarrays
        Averaged_Array[j][i] = np.average(Im[Area_y*j:Area_y*(j+1),Area_x*i:Area_x*(i+1)])

    return Averaged_Array
```

Figure C.4: Functions to window an image about its centre and average over area, used in the sub pixel translate function.

```
def floatToFrac(val, output = False):
    fract = Fraction(val).limit_denominator()
    num = fract.numerator;     den = fract.denominator
    if output:
        string = r'\frac{' + str(num) + r'}{' + str(den) + r'}'
        display(Math(string))
    return num,den
```

Figure C.5: Converts a float value to a fraction expressed as a numerator and denominator.

```
def ImPlot2D3D(img, step=False, ratio=50, cmap=plt.cm.jet, DEBUG=False):

    Z = img[::1, ::1]

    fig = plt.figure(figsize=(14, 7))

    # 2D Plot
    ax1 = fig.add_subplot(1, 2, 1)
    im = ax1.imshow(Z, cmap=cmap)
    plt.colorbar(im) #add a colorbar legend
    ax1.set_title('2D')
    ax1.grid(False)

    # 3D Plot
    if step:
        img = (img.repeat(ratio, axis=0)).repeat(ratio, axis=1)
        Z = img[::1, ::1]

    ax2 = fig.add_subplot(1, 2, 2, projection='3d')
    X, Y = np.mgrid[:Z.shape[0], :Z.shape[1]]
    ax2.plot_surface(X, Y, Z, cmap=cmap)
    ax2.set_title('3D')

    # Scale the ticks back down to original values
    if step:
        ticks_x = ticker.FuncFormatter(
            lambda x, pos: '{0:g}'.format(x / ratio))
        ticks_y = ticker.FuncFormatter(
            lambda y, pos: '{0:g}'.format(y / ratio))

        ax2.xaxis.set_major_formatter(ticks_x)
        ax2.yaxis.set_major_formatter(ticks_y)

    plt.show()
```

Figure C.6: Displays a Image in three dimensional space.

```
def showIm(image, invert=False, disp_dim=100, cmap=None, axes=False, scale=1, grid=False,
grid_interval=1.0):
    if type(image) is np.ndarray: # if image is a numpy array display with matplotlib
        if cmap == None:
            if invert:
                cmap = plt.cm.gist_yarg
            else:
                cmap = plt.cm.gist_gray

        Z = image[::1, ::1]

        sizes = np.shape(Z)
        fig = plt.figure(figsize=(1*scale,1*scale))
        ax=fig.add_subplot(111)
        # Remove whitespace from around the image
        fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
        if not axes: #remove the axes from the plot
            ax.set_axis_off()

        if grid:
            # Set the gridding interval: here we use the major tick interval
            loc = plticker.MultipleLocator(base=grid_interval)
            ax.xaxis.set_major_locator(loc)
            ax.yaxis.set_major_locator(loc)

            # Add the grid
            ax.grid(which='major', axis='both', linestyle='-')

            # Find number of gridsquares in x and y direction
            nx=abs(int(float(ax.get_xlim()[1]-ax.get_xlim()[0])/float(grid_interval)))
            ny=abs(int(float(ax.get_ylim()[1]-ax.get_ylim()[0])/float(grid_interval)))
            # Add the values to the gridsquares
            for j in range(ny):
                y=grid_interval/2+j*grid_interval
                for i in range(nx):
                    x=grid_interval/2.+float(i)*grid_interval
                    ax.text(x,y,'{:d}'.format(image[i][j]),color='r',ha='l',va='l')
        plt.show()
```

Figure C.7: Displays a NumPy Matrix as an image, optionally overlays pixel values.

```
def showIm_PIL(image, invert=False, disp_dim=100, cmap=None, axes=False, scale=1,
grid=False, grid_interval=1.0):
    # if image is actually a PIL Image Object display with jupyter
    if type(image) is PIL.Image.Image:
        if invert:
            image = ImOps.invert(image)
        if image.width < disp_dim and image.height < disp_dim:
            if image.width > image.height:
                scale = int(np.ceil(disp_dim / image.height))
            else:
                scale = int(np.ceil(disp_dim / image.width))

            dims = map(int, ((image.width * scale), (image.height * scale)))
            display(image.resize(dims, 0))
        else:
            display(image)
```

Figure C.8: Displays a PIL object as an image.

Appendix D

Application of the proposed method for out of plane movement

While the proposed method has been proven to be effective for in plane movements this severely limits its application. A more general 3D motion estimation algorithm would be preferable that can measure translations in three dimension as well as rotations. The basic method can be proven to only work for in plane translations as there is simply not enough information output to infer anything else. There are only two values output, the total magnitude of the absolute sums of the differences of the rows and columns.

The method the author proposes involves removing the last step of summing the rows and columns and instead using this the information at this stage to try and investigate motion. Each image has a unique fingerprint and knowing this and how different translations effect an images fingerprint can allow estimation of the translations that have occurred.

To nullify the effect of edges the image will have to be windowed else new information introduced will throw the method off, as it weights centre and edge pixels equally. This can be achieved by cropping out the centre of the image and then zero padding it

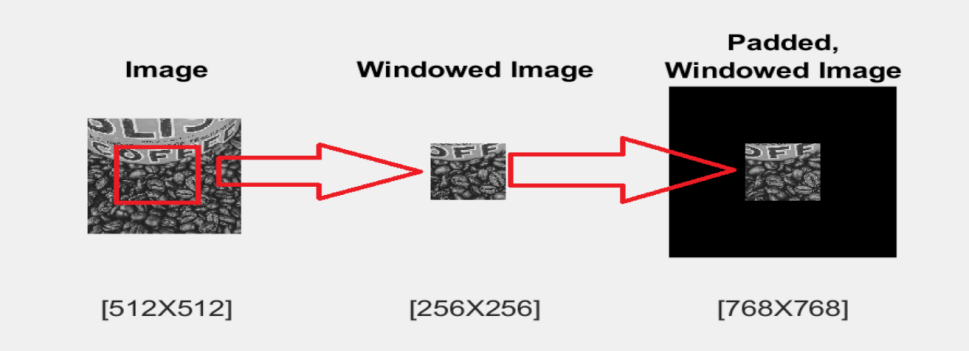


Figure D.1: Windowing and zero padding [18]

D.1 Out of Plane Rotation

Looking at the fingerprint of a simple donut we can identify several features which are affected by an out of plane rotation. Defining the axes x, y, z as being vertical, horizontal and into the page respectively we can observe that a rotation about the y axis causes a compression of the image in the x axis. This is measurable in the magnitude of the sum of rows values. This shift can be described using simple trigonometry, see initial investigation in appendix A.

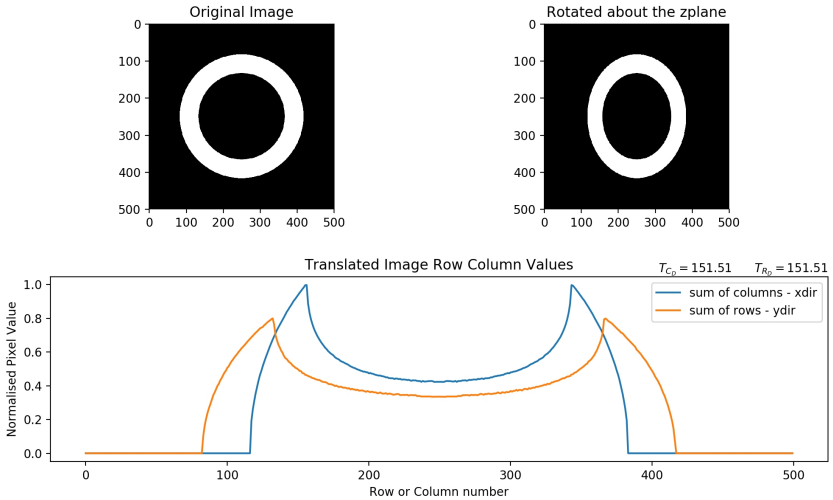


Figure D.2: New Method Rotated out of plane

Quantifying whether the rotation has occurred clockwise or counter clockwise should also be possible using the parallax effect. The side of the image that has rotated closer to the camera will appear larger to the camera and this should be quantifiable using the focal length of the camera and distance away from the observer.

D.2 Movement perpendicular to the Image Plane

Movement perpendicular to the movement plane or more simply movement towards or away from the camera should also be possible to quantify using the fingerprint of the image. Using the example of the image having moved away from the camera, the features describing it have been reduced. In this simple case it is because the edge pixels introduced all have zero intensity values and therefore in a real world capacity this could be more difficult to determine. The percentage with which the image reduces by per unit distance will again be relative to the focal length of the lens system used with the camera. A camera with a shorter focal length system and hence a wider angle lens will demonstrate more of a parallax effect causing the image to reduce faster per unit distance compared to a longer focal length.

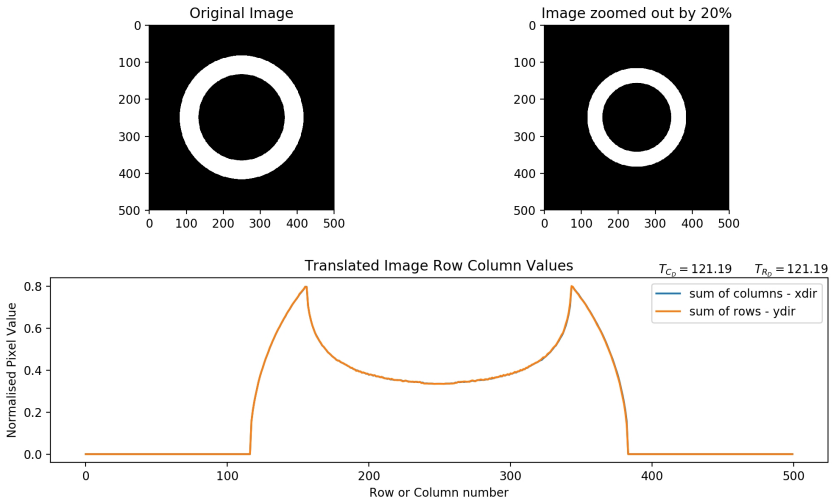


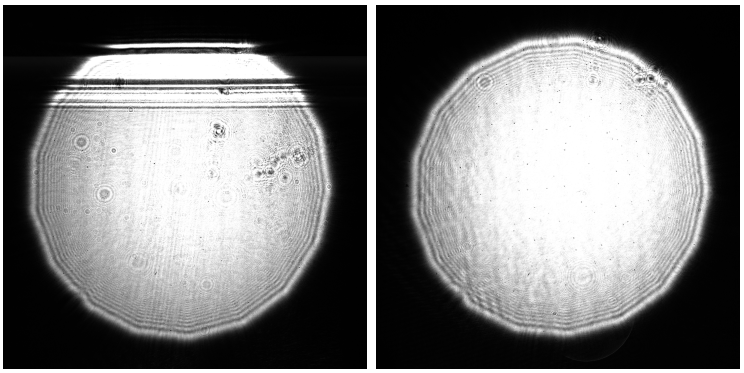
Figure D.3: New Method Zoomed Out 20%

Appendix E

Investigation of Experimental Setup

E.1 Equipment Setup

As part of the proper evaluation of the proposed method it is empirical that it be tested under real world conditions. To understand the test bench developed by Duignan the author worked with PhD Researcher Min Wan to investigate and reduce the impact of any sources of error and noise within the setup including misalignment, dirt, bad focus and fringing.



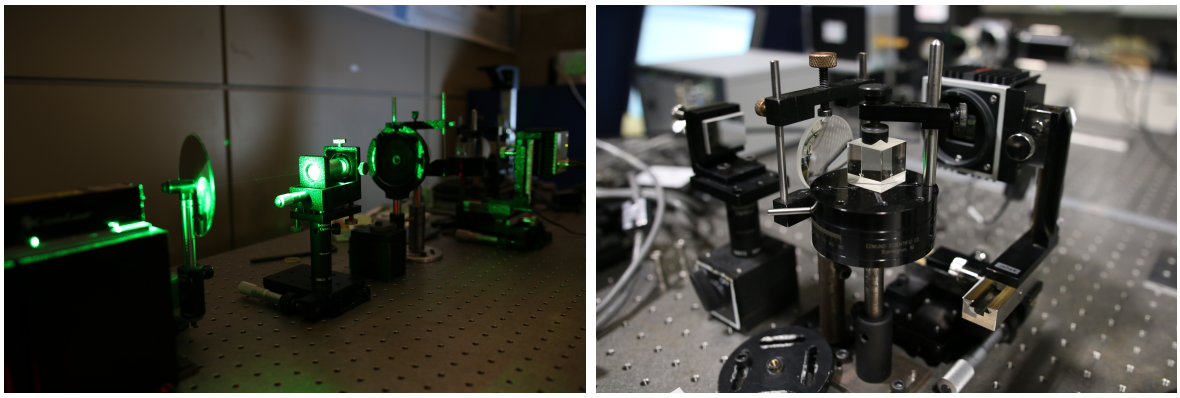
(a) Initial Image (b) Calibrated Image

Figure E.1: Images from test equipment

E.2 Image Collection Automation

One note of Duignan was that the process of recording the images should be automated to enable more samples to be collected and more accurate results. This was investigated and a 3 axis Standa motorised translation stage (*8MT167S-100-28XYZ*) and controller

was determined to be suitable with its 1000 steps per mm accuracy and python interface. Image collection would be automated using a GUI automation tool called PyWinAuto as the software to control the camera did not have a programmable interface.



(a) The experiment in operation (b) Beamsplitter, mirror and camera

Figure E.2: Experimental Setup

E.3 Safety

In order to ensure safe operation within the laboratory it is important the author understand the risk factors associated with this project. The main cause for concern being exposure of the retina to high levels of laser radiation.

In accordance with laser safety standards the maximum permissible exposure limit in Joules per meter squared is calculated using the following equation:

$$MPE = \begin{Bmatrix} 10^2 C_3 C_6 & \forall t > T_2 \\ 17 t^{0.75} C_6 & \forall t < T_2 \end{Bmatrix} \tag{E.1}$$

C_3 is a wavelength-dependant correction factor, and C_6 is a correction factor dependant on the visual angle with which the laser radiation enters the pupil. The exposure duration is denoted by t . T_2 , the exposure duration limit, is given by: $T_2 = 10 \times 10^{0.02(g-550)}$, where $g = \lambda x 10^9$.

Bibliography

[1] *Basic Principles of Image Sensors - Introduction*. http://optique-ingenieur.org/en/courses/OPI

[2] *Bit Depth*. https://www.cambridgeincolour.com/tutorials/bit-depth.htm.

[3] R. W. G. Hunt. *The Reproduction of Colour*. en. Wiley, Oct. 2004. ISBN: 978-0-470-02425-6.

[4] PANTONE. *Pantone Brings the Best to Life.* en. https://www.pantone.com/how-do-we-see-color.

[5] K. Jack. *Video Demystified: A Handbook for the Digital Engineer*. en. Elsevier, Apr. 2011. ISBN: 978-0-08-055395-5.

[6] *Gray_Bit Depth*. en. https://www.azooptics.com/Article.aspx?ArticleID=1151. 2016-12-26T02:15:00.0000000-05:00.

[7] *Phantom V2640*. http://phantom.mx/index.php?option=com_content&view=article&id=241&

[8] N. Wadhwa, J. G. Chen, J. B. Sellon, et al. "Motion Microscopy for Visualizing and Quantifying Small Motions". en. In: *Proceedings of the National Academy of Sciences* 114.44 (Oct. 2017), pp. 11639–11644. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1703715114.

[9] J. Ryle, M. Al-Kalbani, N. Collins, et al. "Compact Portable Ocular Microtremor Sensor: Design, Development and Calibration". In: *Journal of biomedical optics* 14 (Jan. 2009), p. 014021. DOI: 10.1117/1.3083435.

[10] "WikiPhase Correlation.Png". en. In: *Wikipedia* ().

[11] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[12] F. Sylvain, R. Rafael, L. Perrinet, et al. "Sparse Approximation of Images Inspired from the Functional Architecture of the Primary Visual Areas". In: *EURASIP Journal on Advances in Signal Processing* 2007 (Jan. 2007). DOI: 10.1155/2007/90727.

[13] H.-h. Hsu. "Moment-Preserving Edge Detection and Its Application to Image Data Compression". In: 1993. DOI: 10.1117/12.139804.

[14] D. Mas, B. Ferrer, J. T. Sheridan, et al. "Resolution Limits to Object Tracking with Subpixel Accuracy". EN. In: *Optics Letters* 37.23 (Dec. 2012), pp. 4877–4879. ISSN: 1539-4794. DOI: 10.1364/OL.37.004877.

[15] D. Mas, J. Perez, B. Ferrer, et al. "Realistic Limits for Subpixel Movement Detection". EN. In: *Applied Optics* 55.19 (July 2016), pp. 4974–4979. ISSN: 2155-3165. DOI: 10.1364/AO.55.004974.

[16] Q. Sun, Y. Hou, Q. Tan, et al. "A Robust Edge Detection Method with Sub-Pixel Accuracy". In: *Optik* 125.14 (July 2014), pp. 3449–3453. ISSN: 0030-4026. DOI: 10.1016/j.ijleo.2014.02.001.

[17] D. Mas. "Image Based Subpixel Techniques for Movement and Vibration Tracking". en. In: (), p. 8.

[18] C. Duignan. "High Speed High Resolution Digital Image Motion Estimation". en. In: (), p. 102.

[19] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. en. Elsevier, Nov. 2006. ISBN: 978-0-08-047502-8.

[20] *KaySuperComputer*. en. /about/infrastructure/kay.

[21] "Intel Xeon Scalable Platform Product Brief". en. In: (), p. 14.

[22] A. Ltd. *Cortex-A53*. en. https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53.

[23] Cadence. "Xtensa LX6 DPU". en. In: (), p. 13.

[24] R. Dolbeau. "Theoretical Peak FLOPS per Instruction Set: A Tutorial". en. In: *The Journal of Supercomputing* 74.3 (Mar. 2018), pp. 1341–1377. ISSN: 1573-0484. DOI: 10.1007/s11227-017-2177-5.