Real-Time Image Registration Using a Raspberry Pi

Daniel Myers

University of New South Wales at the Australian Defence Force Academy

Image registration is the method of determining the geometric transformation between two related images, each approach to image registration has its specific use, advantages and disadvantages dependent upon the intended application. The Raspberry Pi is a relatively new, cheap and compact hardware platform designed to be a complete, self-contained computer. This project implements an adapted 1 bit image registration technique using Matlab and a Raspberry Pi separately to determine the relative location of the system in real time. This project also assesses the performance of the adapted algorithm and the suitability of the Raspberry Pi for Image registration.

Contents

I. Background 2
   A. Raspberry Pi 2
   B. Programming Languages 2
   C. Peripherals 2
   D. Image Registration 2
   E. Past Research 3

II. Purpose 3
   A. Capabilities of the Raspberry Pi 3
   B. Stand-alone Image Registration Platform 3
   C. Adaptive Low-Complexity (ALC) Algorithm 3

III. System Overview 4
   A. Hardware 4
   B. Software 4

IV. Image Registration Approach 5
   A. Type of Image Registration 5
   B. 1 Bit Adaptive Low-Complexity (ALC) Algorithm 5

V. Method and Implementation 7
   A. Interfacing with the Raspberry Pi 7
   B. Algorithm Implementation 7

VI. Results 8
   A. Performance Testing 8
   B. Accuracy 8
   C. Robustness 9
   D. Speed 9
   E. Camera 10

VII. Conclusion 10

VIII. Future Work 10
   A. Algorithm Improvement 10
   B. Scale and Rotation 11

References 12

APPENDICES

Appendix A. Test Images A1
Appendix B. Progressive Error Images A2
I. Background

A. Raspberry Pi

The Raspberry Pi is a relatively new microcomputer hardware platform (released on 29 Feb 2012) [1] which is a cheap, compact and complete computer. It is developed by the Raspberry Pi Foundation and distributed online mainly by Element-14. The Raspberry Pi comes in two variants, model A and model B. The Model B is the platform used for this project and can be seen in Fig. 1. The model B costs AUD$36 and uses a Broadcom BCM2835 System on Chip (SoC) which contains a 700 MHz ARM1176JFZ-FS Central Processing Unit (CPU), VideoCore IV Graphics Processing Unit (GPU) and 512 MB of Synchronous Dynamic Random Access Memory (SDRAM). Other features include two Universal Serial Buses (USB) 2.0 ports, 10/100 Ethernet 8 Position 8 Contact (8P8C) network adapter, a High Definition Multimedia Interface (HDMI) and RCA video output, Secure Digital (SD) card slot, 3.5 mm audio input, 26-pin General Purpose Input Output (GPIO), Camera Serial Interface (CSI) Buses and currently unsupported Display Serial Interface (DSI). The Raspberry Pi Model B requires 3.5 Watts (W) of power, i.e. 5 Volts (V), 0.7 Amps (A), delivered via a micro USB connection, and weighs approximately 45 grams (g) [2].

Raspberry Pi supports various Linux based Operating Systems (OS). These are typically installed on an SD card which is booted from on power up. For this project the OS Raspbian has been used, which is a modified version of Debian which is optimized for use with the Raspberry Pi [3]. Raspbian boots by default in a terminal screen however a Graphical User Interface (GUI) can be accessed from the terminal or can be set to load on boot.

The Raspberry Pi was chosen for this image registration project due to its low cost, low power requirement and light weight making this platform ideal for mounted systems such as Unmanned Aerial Vehicles (UAV), Autonomous Land Vehicles (ALV) using an on board camera, or other image registration applications such as medical imaging using supplied images.

B. Programming Languages

The main algorithm developed for this project was written and tested using Matlab before implementing it on the Raspberry Pi. Any programming language capable of compiling for an ARM11 processor can be used on the Raspberry Pi, with the main languages used being Python and C [4]. Python is an interpreted language and is the default supported language by Raspberry Pi for its simplicity in learning and debugging [4]. C is a widely used compiled language, which can also be used on the Raspberry Pi. C has been the main language used for this project in order to maximise speed and efficiency.

The primary software package used for the Raspberry Pi implementation of the program is Open Source Computer Vision (OpenCV), a library of programming functions for real-time computer vision in C, C++ and Python interfaces. OpenCV provides an easy way to access, configure and interface with camera peripherals as well as a large range of image processing methods and algorithms.

C. Peripherals

In order to interface with and use the Raspberry Pi for this project many peripherals have been used. Such peripherals include a USB keyboard, mouse and HDMI compatible screen. The most important peripherals in the scope of this project is the image acquisition devices, i.e. the camera. Currently a Logitech C270 webcam is used via USB connection as the camera device, however Logitech does not support Linux (Linux can support Logitech webcams) [5], so control and performance is suboptimum. Ideally, the dedicated camera board designed specifically for the Raspberry Pi would have been used for this project, however due to substantial interface and control issues has not been implemented on this project. This dedicated camera will provide higher resolution, greater frame rates and better software support [6].

D. Image Registration

Image registration is the method of determining the geometrical transformation between one image, the reference, and another related image, the target [7]. This project performs 1 bit image registration on preloaded
and camera acquired images in real-time in order to determine the relative 2D translation of the images. By using this translation data the relative position of the Raspberry Pi system could be tracked or used by an external system. More detail about the specific method and approach of image registration used is detailed in section IV.

E. Past Research

Prior to engaging in this project, past research projects and papers where explored and it was discovered that frames rates of 15-30 frames per second (FPS) and resolutions around 320x240 have been deemed satisfactory by most [8-16] in order to successfully register consecutive related images. Another outcome from these papers was that feature tacking has been shown to be computationally expensive. It also requires extensive modifications to work in real-time on computationally limited systems.

II. Purpose

A. Capabilities of the Raspberry Pi

One of the primary purposes to this project was to test the capabilities and limitations of the Raspberry Pi in order to gauge what level of computing the new platform could be used for. Image registration can be a complex task and traditional methods require more computational power than the Raspberry Pi has to offer. This has been overcome by adapting the registration method and exploring the specific operation of the Raspberry Pi to maximise efficiency.

B. Stand-alone Image Registration Platform

Another purpose to this project was to create a program and system which could be passed a set of related images or requested to capture a series of images. The geometrical transformation the images underwent could be determined and the data output to an external system. This is a ‘black box’ approach to creating a program and system which can be easily implemented with other programs and systems that utilises its own power, processing and camera resources as to not impede on external resources. There have been many platforms used to perform image registration; however there is no documented use of the relatively new platform, Raspberry Pi, being used for this purpose.

The initial aims of this stand-alone system were to:

1. Acquire and process images on-board
   All image acquisition will be conducted on the Raspberry Pi system either using a camera or sourced images with all processing performed on the Raspberry Pi, i.e. external sources are not used for processing.

2. Be completely mobile
   The system will be powered by a mobile power supply, i.e. not connected via mains power, and will be accessed and controlled from a wireless network connection, i.e. no connected network or audio/video cables.

3. Perform image registration in real-time
   The system must be able to perform image registration in real-time, producing an output of relative image translation multiple times every second. As outlined in section I.E, frame rates of 15-30 FPS have proven successful and therefore this is the ideal range for this project.

4. Store the relative translation
   It would be ideal for the system to output the relative image translation to an external source wirelessly to view in real-time, however for this project, storing the output data for later viewing is sufficient. This represents the output being passed to an external navigation, control or other system.

This project has achieved aims 2, 3 and 4 and partly aim 1 due to interface issues and inaccurate slow results when using an on-board camera. Further details can be found under section V.

C. Adaptive Low-Complexity (ALC) Algorithm

The final and important purpose of this project was to develop and test a modified version of the ALC [17]. The purpose of the ALC is to reduce the computational complexity of image registration whilst maintaining accuracy and robustness. This makes the Raspberry Pi offers a good platform to test the algorithm due to its limited computational power and versatility.

A program using this algorithm was developed and tested in Matlab using preloaded and camera acquired images. Implementation onto the Raspberry Pi was then completed using C code with OpenCV.
III. System Overview

A. Hardware

Section I.A discussed the hardware of the Raspberry Pi which comes assembled commercial off the shelf (COTS), however in order to meet the aims of the system detailed in section II.B additional COTS items were acquired and integrated into the system.

In order to meet the aim of mobility a Wi-Pi, a Wireless Local Area Network (WLAN) USB module, was obtained and set up to enable access to the Raspberry Pi wirelessly from a laptop. This is used to edit and execute code and commands on the Raspberry Pi. Since wireless connection was now possible, Secure Shell (SSH) with X11 forwarding could be used to access the Raspberry Pi terminal and/or GUI, replacing the need for a dedicated screen. Another issue with making the system portable was power, therefore a USB phone backup charger was acquired. This charger provides 5 Ampere Hours (Ah) with two 5 V USB ports, one at 0.5 A and the other at 2.1 A. The 2.1 A port was used since the Raspberry Pi requires 0.7 A, therefore an effective battery life of 7 hours (5 Ah / 0.7 A = 7.14 h).

To achieve the aim of capturing camera images on-board, a Logitech C270 Webcam was used via a USB connection. Ideally the Raspberry Pi Camera Board would be used (Fig. 2) however due to interface issues between the camera and OpenCV, has not been implemented.

B. Software

The basic software solution to perform image registration is described by the flowchart diagram in Fig. 3, where the ‘Capture Image’ term is used for preloaded or camera acquired images.

This procedure includes the program set up before getting two images (reference and target) and performing the image registration algorithm which compares the last two images and determines the relative location and stores that data. If the program is not finished, gets a new image and compares it to the last image acquired. Once completed the system shuts down. The current system stores the location data output locally which could be output to an external system.

Section IV.B describes a detailed explanation of the workings of the image registration algorithm.
IV. Image Registration Approach

A. Type of Image Registration

Image registration is widely used in remote sensing, robotic control, medical imaging, computer vision, target recognition and many other uses [7, 18]. There is no general solution for image registration as there are many different methods and approaches, each tailored to suit the needs of the problem [19]. Using a subset of the nine base criteria proposed by Maintz and Viergever [20], this approach to image registration is categorised as follows:

1. Registration Method

An Intensity based registration approach was chosen over a feature based approach due to its reduced computational requirements, aiding real-time processing on a lower end system. The images are processed as 1 bit binary values, i.e. black or white, instead of Red-Green-Blue (RGB) or grayscale, which further reduces the computational requirements as discussed in section IV.B

2. Transforms

Rigid transforms were chosen over non-rigid transformations as a simple starting point for the project. Furthermore this project deals with 2-dimensional (2D) translation between the reference and target images. Again this reduces complexity and computational requirements, however the system is designed in such a way that other rigid and non-rigid transformations may be applied in future development.

3. Modalities

The Raspberry Pi camera board will be the primary capture device, single-modality is used as the Raspberry Pi only has one CSI and using either additional USB webcams or multiple Raspberry Pis would have created unnecessary weight, power usage and complexity for little benefit.

4. Degree of interaction

This Raspberry Pi system is intended for wireless unmanned use, therefore the image registration program was designed be an automatic method, so there is no requirement for human input to confirm the image match.

B. 1 Bit Adaptive Low-Complexity (ALC) Algorithm

Traditional approaches to feature and intensity based image registration reduce the image resolution (i.e. reducing the number of pixels processed) in order to reduce the computational complexity of the algorithms. The ALC differs from this as it reduces the bit depth of the images for processing whilst using full image resolution. Most algorithms use 8 bit grayscale images to register, in this case the bit depth is reduced to 1 bit. Operations such as image multiplication, error calculation and other operations can now be reduced to 1 bit logic operations such as OR, AND and XOR [17]. The results from the registration can then be applied to the full 8 bit grayscale image if required. This section is a summary of the paper on the ALC [17] where a more detailed explanation on equations and processes can be found.

The ALC is based on an optimised Gauss Newton (GN) gradient descent algorithm which uses a process to calculate six motion parameters to describe the rotation, translation and scaling that has occurred between two related images. The relationship between R, the reference image (x and y), I, the target image (x’ and y’) images and motion parameters can be seen in Eq. (1).

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  m_1 & m_2 \\
  m_4 & m_5
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  m_3 \\
  m_6
\end{bmatrix}
\]  

(1)

The basic process of the GN algorithm is to find the motion parameters, which minimise the sum-of-squared differences (SSD) between the target and reference images, Eq. (2) where \( E_i \) is the \( i \)th pixel of E. This can be achieved by using the error between the target and reference image (E=I−R) with the steepest descent images to estimate the vector \( p \), Eq. (3), which is the direction and magnitude the target may move to minimise the SSD. Equation (3) uses the expressions \( \nabla K \) (Eq. (4)) and \( \nabla^2 K \) (the Hessian of K, Eq. (5)) which refer to the first and second partial derivatives respectfully, of the SSD with respect to each motion parameter. By iteratively moving the target by \( p \), recalculating the error and \( p \), the target image will eventually align with the reference image and error will be at a minimum.

\[
K = \sum_{i=1}^{N} E_i^2
\]  

(2)
\[ p = -(∇^2K)^{-1}∇K \]  
\[ \frac{∂K}{∂m_j} = \sum_{i=1}^{N} E_i \frac{∂E_i}{∂m_j} \]  
\[ \frac{∂K}{∂m_j m_k} = \sum_{i=1}^{N} \frac{∂E_i}{∂m_j} \frac{∂E_i}{∂m_k} \]

This project implements a modified version of the ALC which deals with the two translation motion parameters for the x and y translation directions, reducing Eq. (1) to Eq. (6), noting the change in subscript numbering for this particular algorithm.

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} =
\begin{bmatrix}
    x \\
    y
\end{bmatrix} +
\begin{bmatrix}
    m_1 \\
    m_2
\end{bmatrix}
\]

As mentioned previously, by reducing the algorithm from higher bit depths such as 8 bit down to 1 bit images many operations could be simplified. The images were transformed into 1 bit images by finding the mean pixel value of the reference image and applying a threshold using this mean to both the reference and target images so values higher than the mean would be white (1) and values lower would be black (0). The 1 bit version of images such as \( I \) are denoted by \( \tilde{I} \).

The calculation of the error is approximated into two parts:

\[
E^+ = \begin{cases} 
1 & i < \tilde{R} \\
0 & i \geq \tilde{R}
\end{cases} \quad E^- = \begin{cases} 
0 & i < \tilde{R} \\
1 & i \geq \tilde{R}
\end{cases}
\]

The steepest descent images are reduced by using the reference image instead of the target image and are reduced to 1 bit precision:

\[
\frac{∂E_i}{∂m_j} = \begin{cases} 
1 & ∇K
\end{cases}
\]

When calculating \( ∇K \) with 1 bit images, the multiplication operation can be replaced with an XOR operation. Figure 4 shows a comparison of the multiplication operation used for the 8 bit images and the XOR operation for the 1 bit images. Negative results can be represented in 1 bit by a 0 while positive results can be represented by a 1. Note that the multiplication of a negative and positive produces a negative answer whilst an XOR of a 0 and 1 returns a 1, for this reason the results of the XOR operations are inverted.

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>E ( ∇R )</td>
<td>E ( ∇R )</td>
</tr>
<tr>
<td>- -</td>
<td>+</td>
</tr>
<tr>
<td>- +</td>
<td>-</td>
</tr>
<tr>
<td>+ -</td>
<td>-</td>
</tr>
<tr>
<td>+ +</td>
<td>+</td>
</tr>
</tbody>
</table>

\[ ∇K = \sum_{i=1}^{N} E^+_i \oplus \nabla R - \sum_{i=1}^{N} E^-_i \oplus \nabla R \]  

Due to using 1 bit images the Hessian approximation, Eq. (5), is no longer valid and \( p \) is now calculated using Eq. (10)

\[ p = -s \cdot sign(∇K) \]
Where \( s \) is a designated step size, initially a relatively large step size which is halved once the sign of \( p \) reverses consecutively. The algorithm continues until an application specific SSD threshold is met or the maximum number of iterations is reached.

The two main variables for the program implementing the algorithm are the values for the starting step size and maximum allowable iterations were found to have great effect on the results of the algorithm, testing methods to find optimal values are discussed in section VI.A.

V. Method and Implementation

A. Interfacing with the Raspberry Pi

The Raspberry Pi is designed to be a minimalist hardware platform requiring many components to be acquired and setup in order to utilise the hardware. First a 4 Gb SD card was formatted and an image of Raspbian was written to the card, then the mouse, keyboard, monitor and network cable were connected and finally power connected. The OS required extensive updating and setting up. Most the initial work on the Raspberry Pi involved becoming familiar with Linux, customising and setting preferences. Additionally many software packages were required to be installed, such as OpenCV, in order to prepare the device for the project.

B. Algorithm Implementation

The first step to implementing the 1 bit image registration algorithm was done by creating a Matlab program which takes two grayscale images, performs the iterative registration algorithm, and then outputs the two motion parameters in terms of pixel translation. The program was written using the lowest level of Matlab functionality with the intention of being translated and implemented into C code for the Raspberry Pi. Many features unique to Matlab, although more efficient, were not used to allow easy code translation. Higher level programs were created which set up and pass preloaded and translated images to the image registration program so that the exact known translation could be used to verify the output of the image registration program.

The test images used varied in resolution from 256x256 up to 640x480 which can be seen with their corresponding 1 bit images in Appendix A. Appendix B shows the progressive iterations of the tulips image when translated in the x direction by \( T_x = -12 \) and in the y direction by \( T_y = 13 \). Each frame shows the error image, current iteration, motion parameter estimate and the SSD.

Once the program was verified to run successfully, another higher level program was created to access images captured by the Logitech webcam and pass these to the image registration program. The results of this webcam program could not be compared to exact known translations, however the video feed from the webcam was displayed with the program output overlaid as a vector arrow showing that movement over a known distance resulted in the correct direction and magnitude being output.

Using the successful implementation of the algorithm using Matlab, the program was translated into C code using functions from the OpenCV libraries, such as image loading, handling and 1 bit operations. The preloaded image program was run on the Raspberry Pi alongside the Matlab implementation and verified to have identical operation, although much slower which is explain under section VI.

Once the program was verified to operate correctly when compared to the Matlab implementation, code optimisation was conducted to reduce the run time of the program. The initial time required to achieve a successful registration for the tulips image was approximately 5700 ms, as seen in section VI.D this was reduced to an average of 195 ms after optimisation. As done with Matlab, a higher level program was written to operate a webcam and pass captured images to the image registration program, however due to limitations in speed was not seen to run accurately or effectively. All results including an analysis and comparison of speed is detailed in section VI.

This program was desired to run using the Raspberry Pi Camera as the image acquisition device, however interface issues between capturing images and using them with OpenCV prevented the use of this camera.

The performance of the ALC and therefore this implementation of image registration is based upon the reduction of 8 bit images to 1 bit, however C and most CPUs handle data in units of bytes, i.e. 8 bits. This means that 1 bit data is actually stored and passed as bytes, however the reduction to 1 bit does still greatly improve computational speed as matrix image multiplications, such as Eq. (5), are reduced to element wise operations, Eq. (9). Exploration of storing data of eight pixels as one byte is suggested in section VIII, but was not conducted within the scope of this project.
VI. Results

A. Performance Testing

To determine the overall performance of the image registration program three main tests were conducted using multiple preloaded images with different image resolutions. These tests include:

1. **Accuracy** - Are the results as expected

2. **Robustness** - How far images can be translated and still produce a successful registration

3. **Speed** – How long does a successful registration take

This report does not go into detail about the performance of this algorithm as compared to other algorithms as the aim of this project was to implement the ALC on the Raspberry Pi and so alternative algorithms and platforms were not specifically researched. Comparisons of the general ALC algorithm, not conducted under this project, has already been explored [17].

B. Accuracy

In order to test the accuracy of the image registration program, the images were translated by a known amount, passed to the program and then the motion parameter output was compared to the know translation. Since the images are translated by whole pixels and the preloaded images have no noise difference as camera images would, the resulting motion parameters should align the target image to the reference completely resulting in an SSD of 0.

Using a SSD value of 0 as the criteria for a successful registration, multiple replications of the program was run with every x, y combination of translations between -15 and +15 pixels and the successful registrations recorded. This test was conducted for multiple combinations of maximum iterations and starting step sizes with the percent of successful registrations and the average iterations required in order to determine which combination was best.

A wider range of iteration and step size combinations were simulated, however a subset of the most successful combinations have been shown in Fig. 5. The desired result is a combination which has a high success rate and a low average iterations required to register the image.

![Figure 5. Accuracy Results](image)

A value of 16 or 20 maximum iterations with a starting step size of 4 was seen to have the best combination of successful registrations. A Maximum of 16 iterations with a starting step size of 4 was chosen. This is because
having a lower maximum amount of iterations was required in case an image cannot be registered and requires to iterate until the maximum is reached.

Every iteration of the program takes an equal amount of time to complete, therefore to compare the speed of maximum iterations and starting step size combinations, a comparison of the average iterations required to register can be used. Since the program written for Matlab and C on the Raspberry Pi operate identically, these results can be used to examine both Matlab and C. Section VI.C describes the speed comparison of Matlab and C in terms of average total registration time and average iteration time. This can be used with this data to gauge the effects of different settings of maximum iterations and starting step size.

C. Robustness

A test of robustness was desired as it would indicate how far an image can be translated and still result in a successful registration and how many iterations that would take. The images were translated for all combinations of the x and y directions in a range of five pixels, in increasing lots. This was done for 0-4 pixels, 5-9 pixels up until 30-34 pixel where no images could be successfully registered anymore. A starting step size of 4 with 16 maximum iterations was used, these results can be seen in Fig. 6.

![Figure 6. Robustness Results](image)

These results had shown that all images could be successful registered is translated by 10 or less pixels, and most could be registered up to 10-14 pixels, with the tulips image being most successful, registering translations up to 25-29 pixels. By observing what was occurring at these higher translations, it was found that the images were getting ‘stuck’ in local minima where the error images align such that a false direction of \( p \) was resulting. Possible solutions to this were not explored, however are discussed in section VIII.

D. Speed

For both Matlab on a PC and C on the Raspberry Pi, multiple replications of the program were run per image with a random translation in the x and y direction between -15 and 15 pixels. The average time required to register the image, complete an iteration time and number of iterations were recorded. Note the product of the average number of iterations and the average iteration time is less than the average total registration time. This is due to the calculations performed outside the iterating part of the algorithm such as calculating the gradient of the image. The results for Matlab and the Raspberry Pi can be seen in Table 1.
The results showed that the algorithm running on a PC using Matlab could achieve speeds within the desired range of 15+ FPS with resolutions less than 512x512, however the Raspberry Pi with the algorithm written in C code could not achieve these speeds at any of the tested resolutions. Lower resolutions were tested to determine at which point the desired speeds could be achieved, however these images failed to register far more often as can be seen when comparing the performance of the lena image to tulips and lichtenstein.

E. Camera

The webcam was used with the Matlab implemented algorithm, capturing consecutive frames at a resolution of 320x240. Max iterations were set to 16 and the starting step size set to 4. This resulted in an effective frame rate of 23 FPS as any time the algorithm would iterate to the maximum number of iterations, the frame was dropped, therefore the effective frame rate was taken as the actual FPS*Successful Registrations. The actual frame rate was approximately 26 FPS with 89% of frames being successfully registered, 26*0.89≈23.

The speed of the camera on the Raspberry Pi were not recorded as the interface issues between the camera and OpenCV prevented successful registrations.

VII. Conclusion

This engineering project detailed the use of the new Raspberry Pi platform in conjunction with the modified ALC algorithm to perform real time image registration, detailing the capabilities and effectiveness of the Raspberry Pi and algorithm. It was shown that the algorithm was successfully implemented using preloaded images and images from a webcam using Matlab, and that preloaded images were successfully registered. The algorithm showed to be accurate with all images tested when the translation was less than 10 pixels in x and y directions, with speeds within the desired range of 15-30 FPS with preloaded and webcam images. The implementation of the algorithm on the Raspberry Pi in C code was successful for preloaded images however a maximum of only 7.75 FPS were reached, requiring further optimisation of the code.

The system was able to be controlled in a completely mobile state and perform image registration in real time, although not reaching the desired speeds, this project was aimed to get image registration working on a Raspberry Pi and shows great promise for a future system to be refined and implemented on an external systems.

VIII. Future Work

A. Algorithm Improvement

As the algorithm has been shown to work, and within real time for the preloaded images and Matlab webcam images, further improvement to speed optimisation and robustness is required to successfully use a camera with the Raspberry Pi. Pre-filtering the images before registration could be used to prevent the target image getting ‘stuck’ in local minima, once a close result is found the program can revert back to unfiltered images. Another
possible solution to prevent these local minima errors is split the image into multiple section to register each one and combine results for an overall solution, although this would increase computation and complexity.

When using continuous frames such as when the webcam is used, the assumption of constant velocity can be used to estimate the next motion parameters to be equal to those previous. This could reduce the average number of iteration required and overall run time of the program.

As mention previously, the exploration of using a byte to represent eight pixels, rather than the current system of 1 byte per pixel, could possibly enhance the run time of the program. The main concern with this possibility is whether the time and complexity added will be negligible to the outcome.

B. Scale and Rotation

As this implementation of the ALC only includes the two motion parameters which deal with 2D translation, the programs can be expanded to include the addition of scale and rotation motion parameters. If this addition was made, the complexity and run time would increase considerably which would require further speed optimisation of the program, particularly Raspberry Pi implementation.
References