University of Cape Town

EEE4022S

Final Year Research Project

2D to 3D Medical Image Registration

Author:
Myrin Naidoo

Supervisor:
Fred Nicolls

October, 2019

Abstract

Medical images are important diagnostic tools within the field. Many illnesses and conditions can only be diagnosed or found using medical images. This project aims to create a fast and efficient method of locating a 2D scan from within a medical 3D scan. The image should be recognised despite rotation, adjusted colour/contrast and altered scale. The project aims to help future projects that allow for medical imaging segmentation and other image processing on a large scale for automated diagnosis of anomalies and illness. Currently, registration is mostly done from one 2D image to another, without looking at a full scan. Methods such as ORB, FAST and BRIEF are discussed, along with other imaging registration techniques not developed for medical image registration. Anonymity was used with all scans so that the privacy of all patients would not be affected.

Acknowledgements

Thank you to my parents and sister for encouraging me and listening to me complain about my code.

Thank you to Walter, Thomas, Kira and Yusuf for keeping me sane while completing this.

Contents

1 Problem Statement **1**

2 Introduction **1**
2.1 Background . . . 1
2.2 Machine Learning . . . 2
2.3 Image-Match . . . 2
2.4 Feature Recognition . . . 2

3 Literature Review **4**
3.1 Machine Learning . . . 4
3.2 Image-Match . . . 5
3.3 Feature Recognition . . . 5
3.3.1 FAST . . . 5
3.3.2 Harris Corner Detector . . . 6
3.3.3 SIFT . . . 7
3.3.4 SURF . . . 7
3.3.5 BRIEF . . . 8
3.3.6 ORB . . . 8

4 Implementation **10**
4.1 Data Formats . . . 10
4.1.1 DICOM . . . 10
4.1.2 PNG . . . 10
4.2 Machine Learning Iteration . . . 10
4.3 Single-Axis Image-Match . . . 11
4.4 Two Axes Image-Match . . . 12
4.5 FAST, BRIEF, SIFT and SURF . . . 13
4.6 ORB Iteration . . . 13
4.7 Parallel ORB . . . 14

5 Results **15**

6 Discussion **16**
6.1 Match Image - One Axis . . . 16
6.2 Match Image - Two Axes . . . 16
6.3 ORB Serial . . . 17
6.4 ORB Parallel . . . 17
6.5 Timing . . . 17

7 Conclusions **18**

8 Potential for Further Study **19**

9 Appendix A **21**
9.1 Image-Match Dual Axes Code . . . 21
9.2 Result for Single Axis Test . . . 24
9.3 ORB Code from Medium.com . . . 25

10 Appendix B **27**

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This final year project report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof

Name: Myrin Naidoo
Date: 03/10/2019

1 Problem Statement

This project aims to create a fast and efficient method of locating a 2D picture from within a 3D image from an MRI or CT scan. The image should be recognised despite rotation, adjusted colour/contrast and altered scale. Medical imaging and image processing techniques will be investigated. Machine learning is a potential way forward, as it is useful in categorising images and finding patterns. It also has theorised functionality, some of which has been documented but not implemented. The project aims to help future projects that allow for medical imaging segmentation and other image processing on a larger scale for automated diagnosis of anomalies and illness. Currently, registration is mostly done from one 2D image to another without looking at a full scan.

2 Introduction

In the evergrowing field of medical imaging, 3D images seem to be the way forward. Generally, 3D MRI and CT scans are the preferred diagnostic tools over 2D ultrasounds and x-rays, as they show a great deal more detail. When extracting a 2D picture or when comparing 2D pictures, it is important to find where in the 3D model is being referenced. At present, we rely on a doctor's ability to approximately locate the region. This is not reliable, as it is possible that the contrast could be difficult to read or that human error occurs. This project aims to quickly and efficiently find a 2D image in a 3D model.

2.1 Background

Medical images are an important diagnostic tool. Many illnesses and conditions can only be found and diagnosed using this technique. Medical images are predominantly read manually by a clinical radiologist, but this is a postgraduate qualification and South Africa has a shortage of doctors. As shown in the picture below, unless the contrast on the image is correct, it is difficult to tell anything from the scan, let alone locate the image in the body. The image on the right shows a similar scan that has been plotted by Matplotlib. When having a scan like a CT or MRI done, the most expensive component is the radiologist's time, which is spent evaluating the scan.[1] Any automation of this could reduce the cost of the scan and make it more accessible to low-income communities. This could also make it easier to diagnose patients in rural areas where, ideally, scans could be done by a nurse or technician and then processed elsewhere.

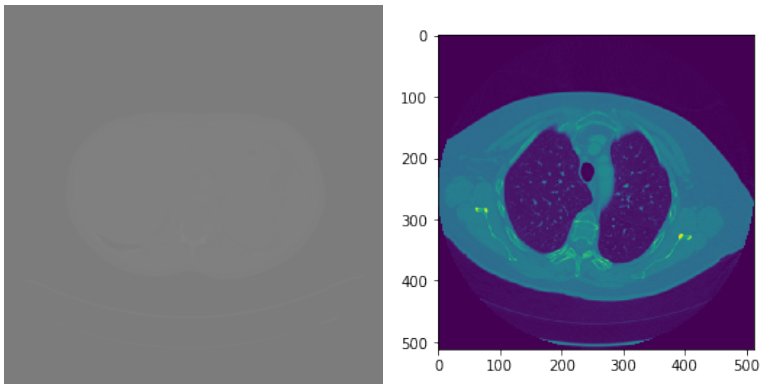

Figure 1: Image with and without correct contrast

Medical image registration can refer to two interconnected processes. One is the locating of a picture and the other is manipulating an image so that it resembles the orientation and scale of another. The focus of this project will be on locating images, specifically in a 3D model or a stack of 2D images. Based on the abundance of articles, it seems that the transformation step seems to be the more researched and

[1] https://www.thestreet.com/lifestyle/health/how-much-does-an-mri-cost-14972340

developed of the two. As the resolution of medical images improves, the size of the data increases, which means it takes longer to find scans. That is why finding an image efficiently is of great importance.

Computer-aided diagnostic is the inevitable progression of medical science. Without it, human error will always be an issue. Using a powerful processor, large numbers of scans can be processed at a time. To identify anomalies in a scan, the image will need to be separated into the anatomical structures. There is no one perfect algorithm that can properly segregate all parts of the human body. In order to run the correct algorithm, the region of the body must be known. Finding the position allows us to run the correct segmentation algorithm. Locating the image in the 3D model can also tell us which structures and organs might be found in this region. Because of its importance in image segmentation, image registration is an important step towards using a computer to completely diagnose illness based on 3D scans. This creates the possibility of 3D scans being taken by minimally trained staff and central processing of all slides by a massively powerful computer.

2.2 Machine Learning

Machine learning can be used to build a simple image categoriser. This falls under supervised learning. A set of images gathered for each category can be used to train a program to label other images within categories. Machine learning can be used to classify the image that is being searched for into whichever anatomical region it belongs. Machine learning has been researched for image registration but not for medical image registration. Image segmentation is the separation of the image into its parts. In the case of medical imaging, it would be to separate muscle groups, organs, lesions or tumours into groups. Registration aligns images to find them in slightly different scans.

Unsupervised learning is a method that allows a computer to note patterns without the need for preexisting labels. A neural network is one solution to this. Unsupervised learning picks interesting points out of unordered data. Most matching algorithms run by finding features from pictures and then matching them. This should have been the starting point for this project. It seemed like an obvious step based on research that stated the applications of machine learning within this sphere. [2]

2.3 Image-Match

The image-match library in Python contains methods for comparing images. The library has a 'normalise difference' function that finds a value between 1 and 0 to see how closely the images match. It can find similar or identical images. It was originally developed to find duplicate images in databases and remove redundancy but, due to its nature, there are many other applications. The first stage that it goes through is reducing the images to grids of signatures. The distances between these signatures are calculated as vectors. These vectors are then compared to evaluate their similarity. This usually involves converting the images to their greyscale values, usually 8bits, and then using these as the signatures. The distance vectors are normalised. The signatures are also referred to as features.

2.4 Feature Recognition

Feature recognition works similarly to the method that image-matching uses. Instead of finding arbitrary points, this method finds specific points to use. There are multiple ways of doing this: the first is corner recognition. Any corners in both pictures are selected and then compared to see if the images match. The next method is feature correspondence using templates, lines, curves or regions to mark areas of recognition. If a template is used, then the centre of it is marked. If a region is used, then the centre of gravity is marked. Using lines as signatures marks the intersection of lines as the signature. Curves use the points of maximum curvature. Techniques for this are discussed in detail in the implementation chapter. The methods include ORB, SURF, SIFT, BRIEF and FAST. These are typical image comparison and registration methods. The advantages of these are that they are orientation and scale invariant. While

[2] https://towardsdatascience.com/introduction-to-unsupervised-learning-8f1b189e9050

accurate, these methods are rather complex computationally, which means that without some change in the algorithm the process will be slow. By incorporating other techniques, the known methods can be improved for larger portions of data.

In appendix A, there is code that is available online and allows one to rotate an image and blur it, then compare it to the original. It is done using a technique called ORB, which will be discussed later. This code was used as a template for the later iterations of the project. ORB is a form of sophisticated feature recognition that is used for realtime computer vision. [3]

[3] https://medium.com/software-incubator/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

3 Literature Review

This literature review aims to address all topics relating to medical image registration within the scope of the project description. The investigation covers techniques that were used and techniques that will be used in the duration of this project. The aim is to get a varied range of techniques and determine their shortcomings. The techniques will be critiqued to see which ones are best to proceed with.

Medical imaging has been slowly automating the process of reading scans and finding anomalies in patients. The two steps of this process are segmentation and registration. Registration is usually only looked at in two dimensions. In current research, the techniques used are looking at images specifically as medical images without looking at methods used to compare other types of images. Existing data sets and models to test software with are few and far between. Of the resources that were acquired, very few are complete. Due to the sensitive nature of some of this data it is difficult to get a hold of. As a result, those not directly in the field, who would like to work towards problems faced, have little data to work with. More studies need to release their complete sets of scans of anonymous patients, and competition should be used to improve techniques. For example, Kaggle hosts competitions that compare the efficiency of different programs to reveal a better algorithm for a set problem. Largely, there needs to be more transparency within the medical field while still maintaining patients' privacy. It appears that most research has been done from a theoretical approach, without attempting any form of implementation. Many have expressed interest in new techniques but have not followed through. It is uncertain if this is due to possessing little technological and programming skill from being in the medical field. It is likely that bio-mechanical engineers and programmers focused on medical sciences are needed to bridge the gap and implement newer ideas in the medical field.

The implementation of most medical imaging registration is not explicitly mentioned in many reports. This is likely due to copyright or patient law with regard to specific companies' intellectual property. Another difficulty with gathering information is that image registration does not relate to only locating or matching an image. Though outside the scope of this project, image registration can mean transforming an image to match the scale and orientation of a similar image. Non-medical image registration is more forthcoming with techniques and practices. Some of these methods, however, are patented. It is also possible that medical imaging uses separate techniques without embracing modern imaging techniques. Based on the material available it appears that, while medical and non-medical image registration use similar concepts, the processing is done differently. [1]

3.1 Machine Learning

Machine learning has been recognised for its image processing applications. It has been viewed favourably for medical images and has shown improvement in the past years. There is some application for improving resolution in images like CT scans, thereby reducing the necessary radiation. It would also improve the quality of scans. Other applications could include searching for an abnormal mass using a picture from another patient as a reference. While there are many potential applications, there does not seem to be much experimentation of these concepts. It is likely because this would affect patient confidentiality. That aside, there are two types of machine learning: supervised and unsupervised. Supervised learning requires massive amounts of categorisation to steer the program in the correct direction. For large data sets like this, it would involve cataloguing and labelling an enormous number of pictures. Unsupervised learning is a far better option: the program finds patterns without needing examples beforehand. In this case, it would track features in the picture (transitions from dark to light or vice versa). Many full scans would be necessary for this. [2]

Most research on machine learning in medical imaging is aimed at image segmentation and not image registration. There is a project by the national institutes of health of the United States of America that has released images for a lesion detection program. The project aims to use machine learning to find lesions using a CT scan. The revolutionary step it made enabled them to detect multiple types of lesions in different regions. There are also projects that are attempting to use machine learning for histology categorisation. By using machine learning, normal and abnormal cells could be identified in much less

time than it would take using current techniques. [3] [4]

3.2 Image-Match

The image-match library is based on a paper called "An image signature for any kind of image" by Wong, Bern and Goldberg. Much like the methods discussed later, their method uses areas of light and dark to determine if the pictures are similar. Squares of pixels are taken from the picture in a grid pattern and are marked by the average of the pixels' darkness. These squares are compared and the distance between them is calculated. It is similar to existing medical imaging techniques that use corners or borders of light and dark to determine if images are similar. This method is fairly robust as it does not require the images to have an abundance of features in order to compare them effectively. It is, however, done in greyscale so there is a chance that some of the information could be lost during conversion. In most cases, this shouldn't be a problem as we are checking to see if the shapes match regardless of colour. Image-match is simple to implement as it has a set library. The command to compare images can be done within three lines of code. It is quick to run and can compare many pictures in a short amount of time. The method is backed by a large database of images, according to the website, but it is not stated how that database interacts with the images being compared. The picture below shows the pattern that image-match generates. [5] [6]

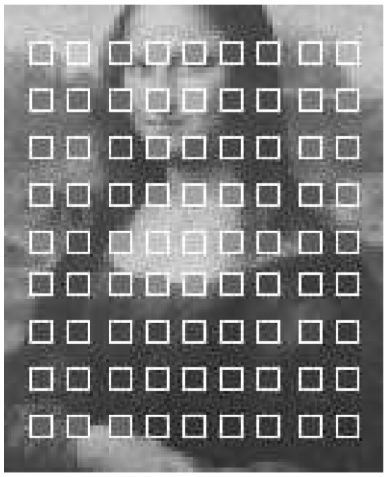

Figure 2: Features Selected with Image-Match [5]

3.3 Feature Recognition

Image registration relies on methods like feature recognition to pick keypoints in both images to establish if they are the same. While this method is more complex, it does offer reliable recognition of images that are in different orientations and scales. In most cases, medical images are in black and white. Thus, methods of image comparison that work by comparing in greyscale have particular benefit. By envisioning the images specifically as medical images, the registration methods become complicated and requires the program to decipher specific features. General image-match programs can use arbitrary points in greyscale to calculate the distance between points and compare the distances in the pictures. Feature recognition seems to be the basis for most image comparison programs. There are many techniques for this, not all of which are equal. Each finds features and uses them in slightly different ways.

3.3.1 FAST

One such corner recognition method is called the Features from Accelerated Segment Test, or FAST. It can be used to track features, specifically corners, and mark them as points of interest. FAST is a very

efficient, high-speed detection method, with a very apt acronym. The algorithm first detects a pixel or point of interest and then sets a threshold value using the colour of that pixel to denote an upper and lower brightness limit. FAST then selects pixels around the point of interest to form a rough circle. If the circle contains continuous groups of pixels that are brighter or dimmer than the threshold, then it is a corner.

This method feeds into machine learning. A non-machine learning version needs to evaluated using a FAST algorithm. The sixteen surrounding pixels must be stored as a vector. The pixels must then be classified as to which are are similar to the threshold, and which are darker or lighter than the threshold [7].

This method is useful in realtime applications, but for medical images it is a bit primitive. The structures in a human body are not angular, so there aren't that many corners to use. However, the algorithm could find processes on bones or be used to find tumours with spiked outcroppings. Sixteen pixels were used in the article for the 'circle' around the selected pixel. If this method was to be applied, it would be prudent to use larger circles to detect larger curves as corners. With this method it may be beneficial to use grid markers instead, like the image-match method does.

3.3.2 Harris Corner Detector

The Harris Corner Detector (HCD) uses a more complex method than the FAST method. It labels a change between bright and dark as an edge. The intersections of two such edges marks a corner. The algorithm first marks a unique square of pixels. Their uniqueness is calculated by shifting the square and seeing how much it changes using the sum of squared differences (SSD). The higher the SSD, the more unique the pixels are. The *E(u,v)* is the sum of all differences in a 3x3 window. The I is intensity and u and v are the x- and [8][9]

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

Figure 3: Harris Method Uniqueness Equation [9]

The above equation has the Taylor expansion applied to it and becomes the below figure to the left. It can then be simplified to the form on the right.

$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \quad E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

Figure 4: HCD Simplified Equation [9]

From there, eigenvectors can be calculated for M to obtain the maximum and minimum values for the SSD (the maximum being the most desirable and minimum the least). From these values, a score R can be calculated. From the R-values, two values can be extracted to determine if the block contains or is near a corner or an edge. The left figure below shows the method of getting the lambda values to evaluate the region. The map on the right shows how to determine what each value denotes. [**?**][10]

This method is far more sophisticated than the FAST method and is computationally more complex. This method also works in greyscale, which is useful and applicable to most medical images. When comparing images, this method uses unique squares and compares them to each other. The idea is that the squares with high uniqueness in two images of the same thing would be very similar. This method can detect rounded corners at high R-values (in figure 5), which essentially means that it can detect certain curves. [6] From a medical point of view, this is immensely significant.

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

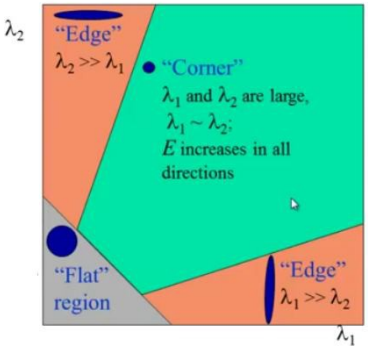

Figure 5: Eigenvalues of Different Conditions [9]

3.3.3 SIFT

Scale-invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) are patented algorithms, so they will not be discussed in as much detail. SIFT is an algorithm for the comparison of images, regardless of their scale or orientation. The success of the two algorithms lead to the development of ORB as a free alternative. SIFT improved on Harris by being scale-invariant. Both SIFT and SURF adjust scale based on an approach using a pyramid structure to keep the images in proportion. SIFT is made up of four steps: scale-space selection, keypoint localisation, orientation assignment, keypoint descriptor and keypoint matching. [11]

Scale-space Selection - Scale-space selection is a measure of the scale at which the picture makes sense. In other words, at which point would it be humanly impossible to tell what it is because it is too big or too small. The increase or decrease in size is done in proportion to the original image and each size is referred to as an octave. Each image is doubled or halved in size. There is a Gaussian smoothing applied as the scale is changed. As the size is increased or decreased there are more images at that size with different smoothing. The differences in the blurred images are then used to find keypoints.

Keypoint Localisation - The keypoints found using the scale-space selection need to be narrowed down to keypoints that are useful. The less useful keypoints are ones that lie on an edge or have unsatisfactory contrast. A Taylor series expansion of location extremes is used and then a 2x2 Hessian matrix computes the principal curve. This results in an equation that can be used to test the keypoints to see if they are useful.

Orientation Assignment - A histogram is generated based on the direction of keypoints from each other. This is used to assign an orientation.

Keypoint Descriptor - The keypoint descriptor labels the features of interest. A 16x16 window is made around the point, and is then divided into 4x4 blocks. An orientation histogram is made for each of the smaller windows. The 4x4 blocks make up eight orientation vectors that result in 128 dimensional descriptor values. Rotation and illumination dependence still need to be addressed.

Keypoint Matching - The images are matched by the distance between features to eliminate false matches. [11] [12]

3.3.4 SURF

The SURF algorithm is for rapid invariant representation and comparison of images. It has two steps: feature extraction and feature description.

Feature Extraction - Feature extraction is done using a Hessian matrix assigned to each pixel. A determinant is needed for the location and scale. Using this, the scale can be changed to any value. It uses an approximate second-order Gaussian filter to evaluate an image with integrals very quickly. The

scale-space is calculated using the same method of octaves as SIFT but without the Gaussian blur. The documentation of this is slightly unclear because the mathematics involved are advanced. It will not be examined further as this method is patented and so was not used in this project.

Feature Description - The feature recognition first calculates the x- and y-direction with a radius of six pixels around the feature. The sum of this is changed by pi/3 to find the main orientation. The descriptor for this feature is stored in a 128-bit vector. [13] [14]

3.3.5 BRIEF

BRIEF stands for Binary Robust Independent Elementary Features. The first step of this algorithm is to smooth the image using Gaussian smoothing. This reduces noise and makes edges easier to find. The algorithm picks a set of pixels, x and y, and compares them. If pixel x is greater than or equal to pixel y in intensity then the result is one. If pixel x has a lower intensity than pixel y, it is zero. Pairs are chosen through one of many methods. Each image patch has a unique identifier in the form of a feature vector, which is a 128- to 512-bit string. The vector can be represented by the equation below. This method recognises images easily but does not allow for rotations within the plane. [15]

$$f_{n_d}(\mathbf{p}) := \sum_{1 \le i \le n_d} 2^{i-1} \, \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

Figure 6: Formula Followed by BRIEF Vectors

3.3.6 ORB

ORB stands for Oriented FAST and Rotated BRIEF. It is a free-to-use algorithm that was made to supersede SIFT and SURF, which are patented. It was developed in 2011 by four developers in Willow Garage, California. ORB combines the quick orienting of FAST with the rapid computation of BRIEF. Thanks to FAST, the image is quickly set to the correct orientation for the image it is being compared to. BRIEF checks the similarity. As can be seen below in figure 7, ORB finds hundreds of features in a picture. [16] [17]

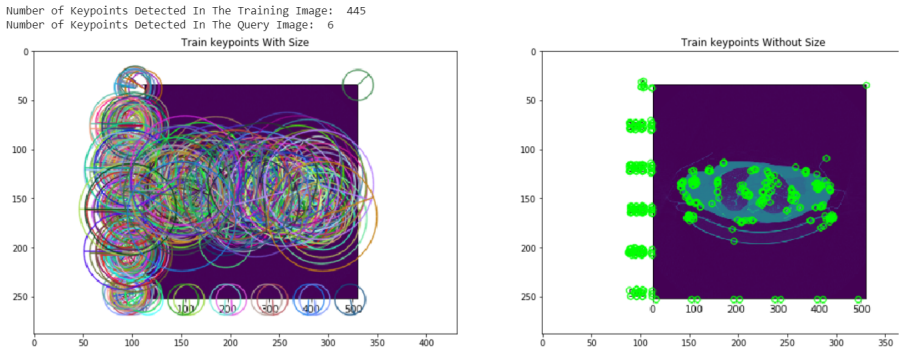

Figure 7: Features Detected with ORB

The implementation of most medical imaging registration is not explicitly mentioned in many reports. This is likely due to copyright or patient law with regard to specific companies' intellectual property.

It is interesting in that it was created for realtime computer vision. Because it was created for realtime use, it is very fast. The aim of it was to use multiple cameras and quick registration between them

without synchronising the orientation or frame. Image registration is used to match features between the cameras and see where they are in relation to each other.

4 Implementation

The code was implemented using Python 3 notebooks on Google Colaboratory[1]. Separate notebooks were made for each iteration of the program. There is some redundancy but it was useful to go between the versions while adding to the subsequent iteration. While GitHub and Google Colab were both used for version control, this method proved easier when viewing multiple versions at once to rework components that were not used in the later iterations. All versions were executed using the GPU on this website. The later versions of code required using a high RAM mode. Some short code snippets are included in the text with longer ones in Appendix A. The code is all online to be viewed.

There is a link to different versions of the code created at: https://github.com/AsthmaticDonkey/Thesis

4.1 Data Formats

The first component that needed to be addressed was which data format to use. DICOM files are common for medical scans and can house an entire 3D scan. The scans could be interpreted as PNG or JPEG images but these are much larger files and would take a considerable amount of time to read into a program. The benefit of the DICOM files is the metadata contained in them. The DICOM file contains information, like the size of the array, embedded into the file, so separate functions to find this are not necessary. DICOM format is also a lot smaller than a full scan would be in PNG. DICOM files and PNG files were both used for this project as they each have their benefits. However, the conversion between the two types takes a great deal of time. Images from the DICOM file cannot be automatically converted to the same format and colour space as the PNG files.[2] The PNG files use the RGB format and the DICOM seem to be in some form of greyscale. No conversion library in Python could convert the DICOM pixel array to RGB.

4.1.1 DICOM

The DICOM file houses a pixel array variable which contains the image data. It can be manipulated exactly like a normal array in Python. The data is supposedly stored in YCbCr format. When treated as YCbCr, it did not convert to other formats. Using Matplotlib, one can display the values with the correct contrast but not with the CV2 library. The Pydicom library was used to interact with these files, which were used for the image-match versions and, to some extent, in the feature recognition (ORB) versions.

4.1.2 PNG

PNG are very versatile image files but they are big and take long to load into memory. They can be read by the Matplotlib or the CV2 library. They are stored in RGB format and are easily converted to other formats. Most image manipulation software and libraries can work with these files. These were used for the machine learning versions and the feature recognition (ORB) versions.

4.2 Machine Learning Iteration

The machine learning version of the code was interesting but failed after many attempts and a considerable amount of the time allocated to this project. The Fastai library and online course were used for this. Their library should have been relatively easy to implement as there are step-by-step instructions in their course notes and Python notebooks. Alas, this was not to be. For this iteration, the code trained a machine learning algorithm with pictures of CT scans that had been separated into categories of head, neck, thorax, abdomen, pelvis, arms and legs.

[1] https://colab.research.google.com

[2] Colour spaces are which set of numbers represent a colour or gradient; i.e. RGB (red, green and blue)

The initial attempt at this was to use images from the NIH Clinical Center using scans of the thorax, abdomen and pelvis. [3] This dataset has thousands of CT scans that contain lesions. The dataset was developed to use machine learning to detect lesions, specifically. The first issue with this approach was that the scans weren't completely humanly readable. The images needed to be adjusted by contrast. They were saved in PNG format to save space but this meant that each image needed a value subtracted from its pixels before it could be deciphered. Even in PNG format the images were over 100GB.

The second problem with this was that the scans were not labelled. As I have some background in anatomy, I attempted to manually label and sort these images. This quickly became quite laborious without much progression as the Fastai library required about 200 images. There were also some issues with the machine learning libraries and existing medical image processing libraries. It became clear that some of these libraries could not both be installed within a single environment. Machine learning seemed like the way forward, so that was attempted.

After sorting some of the images and discovering how long it would take to use this method, I tried a different approach. The Fastai library uses a JavaScript command in a browser to download batches of images from search engines like Google. Using this to acquire scans failed too. After managing to word the search so that it was not inundated with cartoons and other irrelevant images, I attempted to train the model to sort the data into categories. One of the steps in the process of training the model is to remove images that are similar to images in the other categories and then delete any that would confuse the process. Unfortunately, CT scans of different regions were determined to be too similar, so the data set as a whole was removed and there was nothing to verify. When run with the few images from the NIH dataset, the result was the same. At this point, it became apparent that this method was not working or at least should not be the first version.

It would theoretically be possible to use the image-match program to make a rudimentary categoriser and feed the sorted images from that into a supervised machine learning algorithm. The process could be repeated using the now-developed machine learning categoriser to eliminate some error from the image-match version. The confounding slides could then be removed and leave categorised data for a supervised machine learning algorithm. An unsupervised machine learning process could be applied to the categorised data to find features. By categorising the slides first, there are fewer differences between them so the unique features would be reduced. The method could also be made parallel using a thread for each category. Rotated and inverted slides should be included in the data to prevent errors in the slide from causing a false classification.

4.3 Single-Axis Image-Match

This version of the code used an image-match library. The 3D image is represented by a 3D array. By using 2D 'slices' (or slides) of the image, it can be compared to the 2D image that is being used for registration. This step used a simple for loop structure that iterated through the slides, finding the image that is closest to the one being searched for. The program searches vertically through the images and compares horizontal slices.

As discussed earlier, this version uses a library that takes a grid of blocks to use as features. The versions using this method are not using feature detection, which involves finding elements like corners to use as a point of recognition. This program creates its own features by taking the average of the blocks in the grid it creates. The algorithm behind this is simplistic and quick. By not searching for features, this algorithm maintains a low computational complexity.

The model assumes that the scans are divided into the horizontal or near horizontal. A near horizontal section would return the closest match, so it would still be in the correct region but would not perfectly locate it in the 3D image. Despite its simplicity, this version could be usable, as these images are often only used within this plane. Even if the images are a few degrees inclined from the horizontal, the program should still be able to find the slice as it looks for an image that is most like the image being searched for, as opposed to an exact match. Although simple, this is a fast and accurate version of the program.

[3] https://www.nih.gov/news-events/news-releases/nih-clinical-center-releases-dataset-32000-ct-images

It could be applied in circumstances where the orientation of the slide is definitely known. Scans like this are usually in a single position and orientation so it is likely that this would be the case.

Below is the code for comparing an image with the pixel array of a DICOM file. The values are stored for sorting and establishing the closest image at a later stage. This shows the image signature and the normalised distance being calculated. It iterates through the z-axis of the DICOM images, meaning it takes horizontal sections from head to foot.

```
[ ]  1 inti = 0
     2 mlist = []
     3 direction = 0
     4
     5 gis = ImageSignature()
     6 a = gis.generate_signature(img)
     7 directory = '/content/drive/My Drive/ThesisData/'
     8
     9 for inum in range(0, dicom_image_file_0.pixel_array.shape[0]):
    10   b = gis.generate_signature(dicom_image_file_0.pixel_array[inum])
    11   mlist.append([gis.normalized_distance(a, b),inum,direction])
    12   if(gis.normalized_distance(a, b) == 0):
    13     print('found')
    14     break;
    15
    16
```

Figure 8: Features Detected with ORB

4.4 Two Axes Image-Match

The two axes version works similarly to the single axis version but it searches through the 3D image vertically and then horizontally. It takes the image from the two dimensions that are the closest to the location of the 2D scan. This version first creates a new array of images by taking rows from each slice, which are examined using the single axis method, and then turns them into vertical images slides. The vertical slides are then searched through much in the same way the single axis slides are searched through.

There are two versions of the two axes method. The one uses image files from a directory and sorts them into a 3D image. The other version uses a DICOM file, which is a medical imaging format that holds a 2D or 3D array of pixels, along with metadata about the patient that the scan belongs to (like age, height, weight and sex). As these slides have been anonymised, much of the metadata has been deleted. The pixel array can be sorted through and split into horizontal images, which must be converted to vertical images because of the structure of the array. The converted slices are much the same as the horizontal slices and can be compared in the same way. The data proved difficult to split into horizontal slices in a different direction through the x-axis. It would involve manually forming each row, pixel by pixel. The program timed out on the online Python notebook. It is likely that a multithreaded approach is needed in order to accomplish this.

This method was easy to implement and it is fast at comparing images. It would be suitable alone for basic registration. A diagonal axis may be added for better searches along axes that are unconventional for medical images. This program can find slices that are on an angle but the registration becomes a rougher estimate as the angle increases. The x-axis was able to locate an image that was at 45 degrees. The program will always return a closest match. This closest match would be, at the very least, in the correct region of the body, so segmentation would be possible.

A version of the code to do this search is in Appendix A. The code searches for slide 350, but this number can be changed to search for a different slide. This same code was put into a loop to search for multiple scans for testing purposes. The code is fairly simple and can be altered for other applications.

4.5 FAST, BRIEF, SIFT and SURF

These approaches were not implemented during this project but they were researched thoroughly before the implementation of ORB was done. FAST and BRIEF form the basis for ORB, so their code is incorporated into that version's. FAST does not account for rotation and so it was not used by itself. SIFT and SURF are both patented so they were not used either. SURF and SIFT are both older techniques that have not been improved on. ORB was therefore the newest and best choice for the next generation of the code.

4.6 ORB Iteration

ORB is a form of feature recognition that does not rely on scale or orientation. In the previous iterations, the orientation needed to be correct in order for matches to be detected. The ORB method uses hundreds of features, only a few of which would need to be matched. For this portion, a version of code for ORB was used as a template and then converted to allow for medical images and a 3D image to be searched through. A version of the code can be seen in Appendix A. The code rotates the image, blurs it and then matches it to the original image. The CV2 library was used for the implementation of ORB. Code for FAST, BRIEF, SIFT and SURF were also examined on this site to compare the processes with that of ORB. [4]

This method used a training and a testing image. The training image is what is being located in the 3D image. The testing image is a slice of the 3D image. It is likely that there will be some features that match on the incorrect slices as there is a small difference between some slides. To counteract this, the program will eventually select the image that has the closest corresponding features. The Gaussian smoothing on the image was reduced because the images are low noise and a higher number of recognised features is preferable. Some Gaussian blur was necessary to avoid matching random pixels. A version of this code involved plotting and saving the pictures using Matplotlib to convert the images to colour. The image was then converted to RGB and greyscale for processing.

This still involves searching through thousands of images. A potential way to speed this up would be to search every tenth image or so and then fine-tune the results after an initial search. Otherwise, the best approach may be to multithread the program. The resources are easily split up, so it would make sense to use a large number of threads with GPU execution. By using multithreaded code, the images can be converted to PNG images, read back in and then compared. Creating and loading the pictures back in took an extremely long time to do because one body has 3300 horizontal slices that need to be converted to pictures. The vertical axis would have another 512 slices and nearly 3300 for any diagonals.

Another version of this same iteration used a DICOM file pixel array to accomplish the same. It was problematic to convert the images from the DICOM file into an RGB version that was compatible with the code. This was remedied by plotting the images using Matplotlib and saving them as PNG files before loading them back in as images. This seems like a laborious step, but it was seemingly impossible to convert from the current colour space to RGB or greyscale. Due to the high number of images there are in a DICOM file, it was not feasible to convert all the pictures. Instead, the image-match library was run to get a rough estimate of where the slice is. The result from the image-match was then fed into the ORB code to fine-tune the location of the slice. The ORB method iterates over 50 slices to either side of the slice found by the image-match library. Even if the image-match cannot find a clear answer, the ORB portion of the code will find it based on the image-match's best guess.

Interestingly, during testing it was discovered that the slices all have about the same amount of matches so this was not a clear indicator of which slice was the correct one. However, each feature match has a distance associated with it. This distance is a clear indication of how similar the images really are. The sum of the distances was calculated and the image with the lowest sum of distances was thus the match. This produced consistent answers but the slice returned was slightly off from what it was supposed to be. The returned slice was one away from where it should have been and could be seen when compared.

[4] https://github.com/deepanshut041/feature-detection/blob/master/orb/orb.ipynb

After this, the average distance of the matches was taken and compared. As the number of matches does not respond to it being closer, the average is a better estimate. The sum of the distances was divided by the number of matches and the lowest of these was taken as the match.

4.7 Parallel ORB

To circumvent the exorbitant amount of time converting the slices took, the code for this was made parallel. The code was executed on an online GPU. It used one thread for every 200 slices, so for the almost 3400 slices it used seventeen threads. The images were saved by these threads and then read back in by another seventeen threads. The vertical slices were created and saved by four threads. As there were only 512 images, fewer threads were necessary. At this point, RAM became scarce as all 3400 images were loaded into an array. The high powered settings on the website helped with this but RAM was still a slight issue. Luckily, at this point, not much else was highly dependent on RAM. The program in question simulates 12GB of RAM. After being read back in, the images are now in RGB format and can be input into the ORB programming. The ORB programming can now work with all the images.

The conversion to greyscale and the Gaussian blur needed this to function. For this iteration, the Gaussian blur was not used. With the blur on, it seemed that there was not enough variation in the images and the distribution of the matches was seemingly random. In most cases, when the ORB technique is applied the blur stops random pixels from being matched. In this image set, there are a lot of very similar images. When the blur is applied, you lose many of the features in the picture. The features lost made the program incapable of functioning properly to differentiate between the images.

The comparison would still be done in serial but could be changed to parallel because the number of axes increases or, in the future, once scans get better resolution. As the resolution increases, the amount of data stored increases as well. With these changes and the parallel implementation of reading the scans, the model is now fairly fast and orientation- and scale-invariant. It should now be able to find a scan regardless if it has been altered.

This is the most scalable version of the code. The numbers of threads can be increased and larger amounts of data can be scanned in. It is uncertain how it would handle more data within the current system as it does use a lot of RAM. The program has crashed on Google Colab due to the RAM requirements when more data was used. There was an attempt to make the actual ORB code parallel but that crashed too frequently. It was attempted to make it use the same number of threads as the parallel reading and writing methods but it crashed with an unknown error. It is likely due to the RAM requirements.

5 Results

The various versions were all verified by checking which slide number was returned and which was searched for. Each version was also verified by visually examining the target and resultant images. The resultant images were also compared to the slides next to the one of interest to ensure that it was not difficult to tell if the image is correct visually. For the angled slides, there was no confirmation using the number of the slide so it was only confirmed visually by plotting the images side by side. The methods used were known and proven for 2D applications.

At this point the various versions work in two axes. The first test is to check at what angle away from the axis the slide could still be recognised. Measuring the angle is a simple matter of trigonometry. Side A of the triangle below is the number of slices the composite image stretches over and side B is 512, which is how many pixels wide it is. A slice is considered as being one pixel thick for the purposes of this test. The angle is then tan inverse of side B over side A. The aim of this test is to see to what degree a scan will still be found and how close the approximation is. The test was repeated with adjusted components to test the vertical axis search. The time taken for the execution of each version was taken to compare the time constraints of the various models.

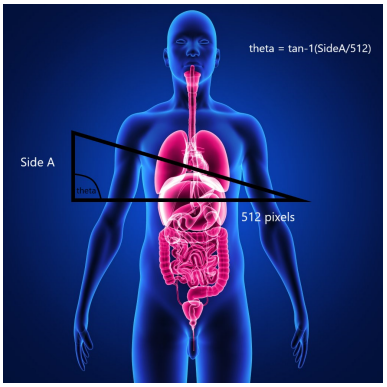

Figure 9: Calculation of the Angle for First Test

The test to determine at which angles the program is successful was run several times per version of code and deemed a failure if the result had not been used to make the composite scan. The angles ranged from 0 to 45 degrees. The samples used were slice 350 to 862 of over 3000 slices. The composites started from slice 350 and gradually changed depending on the gradient in question. The composite slices composed mostly of the heart and lungs at varying levels with the exception of the 45 degree composite that included some of what appears to be the liver. A composite scan with the original at slice 350 can be seen in Appendix A. The match image program returned consistent results with regard to which slice it found and the percentage match found. Below, a slice taken at an angle can be seen with the slice that the program found. They are nearly identical. The left slice was taken at an angle of 26.57 degrees. Even at this incline, the returned slice is adequately similar.

To run the full test, including loading images and testing for all angles, the single axis code took 267.63 seconds. The two axes version took 296.57 seconds with the same tests and data. Single axis ORB took 304.98 seconds.

For the parallel version, the reading and writing of the images were measured first, separately. This alone took 452.27 seconds. This is already higher than the other two variants but the ORB code detects similarities with less constraints than that of the other versions. In theory, the ORB code is superior to the image-match code. The sacrificed time is specific to starting up the scan. Multiple slices can then be located using the ORB code without reloading all of the data. It took 94.08 seconds to locate a single scan. The test set was run on this same data set but it took 1274.14 seconds to run on this version. This was on 13 images. Collectively it took 1726.41 seconds.

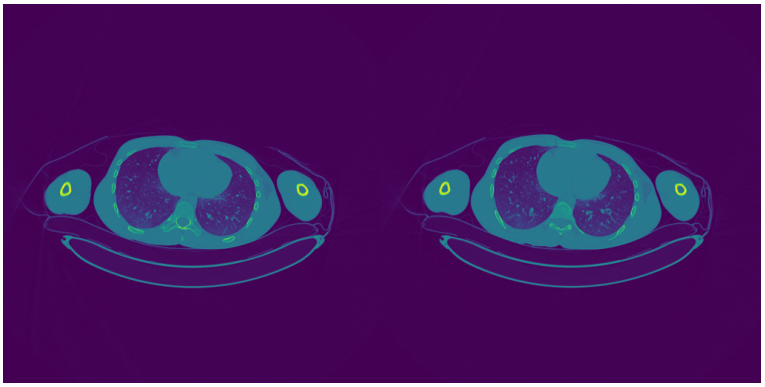

Figure 10: Slice at Angle (left) and Found Slice (right)

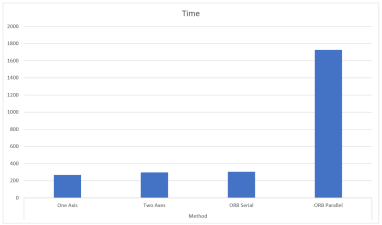

Figure 11: Bar Graph Showing Execution Time

6 Discussion

6.1 Match Image - One Axis

While it is limited by orientation, the quickest algorithm is the image-match. This may be because it can work with the DICOM file without any conversion. It is the simplest of the interactions and can quickly find an image, provided it is in the right orientation. It can also detect images that are at a slight angle. Surprisingly, with the first test, using the image-match library and a single axis scan, the results were fairly accurate. The slices were all within a small range of the slices used to create the composite. Within this library, anything less than 0.4 is seen as a probable match, with zero being a perfect match. Most of the results were approximately 0.3 or below. At this resolution of registration, it would be possible to determine which segmentation algorithm is appropriate. At this point, further angles could have been tested but it becomes superfluous given the scope of this study. Even though it did find a match for the 45 degree slice, it would be prudent to only use it for slight angles as there is too large an interval in which a match could be found when the angle is increased. On repeated tests, it was seen that the program delivered the same results every time.

6.2 Match Image - Two Axes

This code is predominantly the same as the single axis version, with the exception being that it was slower because it scanned the second axis. A large portion of the runtime was spent simply recreating the picture array so that there were 2D portions in another direction. The scanning of these new images would have been at the same rate as the 1D scan but there are fewer images in the vertical set than the horizontal (as the array now holds 512 larger images). This method is fast and can theoretically find a match for 45 degrees from each axis. This should almost always result in a correct slide. It did not return an incorrect answer during the tests. This version balances search orientations and speed. The second axis adds a degree of reliability with angled searches. With the addition of a rotation matrix on the image being searched for, this could be a valuable tool that can be built on.

6.3 ORB Serial

The serial ORB code was executed similarly to the above attempts. The previous version was incorporated into this new code, so it was slightly slower. It found the correct slide in several tests. The first half that located it is orientation variant, so the program as a whole will not discover rotated slides or it may pick up rotated slides inconsistently. The matching algorithm finds a slide within 50 of the correct match. This version does not add much to the image-match. While it does double-check the single axis version (and checks for rotation), it will not find anything if the image-match is not similar enough to the correct slide.

6.4 ORB Parallel

This version returns the correct result consistently. It did not include the image-match library and ran solely on the ORB method. The program initially failed to find the slide when the Gaussian blur was implemented. Once the blur was removed completely, the correct scan was consistently found. This model used the number of matches found and not the average distance like previous models. With the blur, the images appeared too similar and there was a very similar number of matches and average distance. With the blur removed, the images differed greatly in number of matches and distance.

6.5 Timing

The timing between the single axis and dual axes methods are fairly similar. There was just a few seconds discrepancy but it is negligible. The serial ORB produced a similar result. While it does take longer, the ORB technique can pick up if the scan has been rotated or scaled, so it would be prudent to use this technique over the others. This algorithm used the single axis method and then used ORB on 50 slices on either side of the one chosen by the single axis image-match. It took several hours to run this ORB code on the entire data set, so this was the compromise. It is still more likely to find rotated slides than the single axis or double axes methods. Rotated versions of the full scan could be added as an extra axis or a rotation matrix could be applied to the searching image. The rotated version would be searched for as well.

The parallel version took the longest of all the programs. This was due to it converting the entire data set into PNG format so that it could be read by the ORB code and the CV2 library. Even though this was the slowest method, the algorithm searched through the slices in the most ways due to the images' orientation being inconsequential to it. The program took an excessively long time to complete the thirteen registrations, especially when compared with the other versions. It could possibly be improved by multithreading the ORB method but it would be faster to use the image-match version and scan using the original and rotated versions of the image. The RAM and computing power needs for this version are absurdly high, so this may only be a viable option in the future with higher computational power. Implementing a machine learning approach with ORB could speed up the process.

7 Conclusions

The single axis search is a good first step but the resultant location needs to be further vetted if a completely accurate answer is needed, especially if the scan is angled. The search is, however, very fast. It can be used as the basis for further methods or it can be incorporated into other algorithms. Considering just the results, image-match is the best version to use if the slide has been correctly oriented. It is fast and produces reliable results. The biggest attraction to this method is that it can compare images in the original DICOM format. It drastically reduces the time needed (as opposed to the ORB version that converts the images first). It can locate images to 45 degrees from each axis. This is the best version at present. It can be improved on but it locates a scan quickly, efficiently and accurately.

The parallel ORB code is the most able to handle large amounts of data as it is highly scalable. When run on a GPU, it can have an excessive number of threads. It is slowed by the conversion of the images but more threads could be used to reduce this in larger data sets. One thread per 200 slides was the minimum that seemed to execute within a tolerable time frame: a thread per 100 slides is recommended, if possible. The parallel ORB method was the most in line with what the project description intended but it is much too slow to be used for anything more than one scan being searched for in a single 3D scan.

Ultimately, the image-match version is the best method to come out of this project. It can handle big datasets within a short amount of time. It definitely has room to be improved on, as there are many small steps that can be taken to deliver a better result. It allows for big data levels of scans to be searched through for image registration. The approach of using a grid as features is a far faster system than any of the methods that tried to find existing features in pictures. This method could also be made parallel to increase the speed for larger 3D scans or those with better resolution.

8 Potential for Further Study

The first avenue for further study would be to apply machine learning concepts to the project. As previously stated, this was unsuccessfully attempted but it is definitely possible. Unsupervised learning is likely the way forward, and not supervised learning for categorization. Machine learning could provide shorter search times and more accurate results. The use of machine learning could also allow for more axes to be search on for more accurate results for slides that do not go through strictly vertical or horizontal scans.

The program should be coded in parallel from the start. The data is easily separated and computed without communication of threads, so resource division would be a good method to use. As many threads as feasible should be used and the code should be executed using a GPU to optimize the speed. It was shown that 100 slices could be processed very quickly, so it is suggested that around 33 threads are used for scans with 3300 slices. Double this for multiple axes searches.

The next step of this project could be the other form of registration or segmentation. Registration can also be transforming an image so that the scale and orientation matches the image it is being compared to. Segmentation is the division of a medical image into the anatomical components. This could involve simply separating the slices into muscles, bones, fat and specific organs. Alternatively, it could take the more complex route of labelling specific muscle groups and naming bones, portions of organs and nerves in an MRI. Segmentation is common in 2D but less so in 3D, although 3D segmentation could allow for better visualisation and/or 3D printing areas of damage or tumours.

With proper categorisation and registration it would be easy for anyone to read a scan, even without much training. Machine learning could then improve this process by detecting anomalies without human interaction. Human error due to exhaustion and negligence causes many deaths in the medical field. The more of this that is automated, the better for healthcare. Automatic detection could be highly beneficial on a microscopic level, when MRI scans are capable of this resolution, for histopathology purposes. By automating scans down to this level, it would be possible to see microscopic tumours or cellular irregularities.

The categorisation would be done more easily by finding the closest match on a slice from a known region of the body. It would take minimal time and the coding for this would be extremely simple. Categorisation is important for segmentation, as no one algorithm can separate all types of tissue. Algorithms are specific to regions of the body such as abdomen, thorax or pelvis.

For the ORB code, the longest component to execute was the conversion from DICOM to PNG format. It would be possible to store the raw scans as image files instead of in a DICOM file. This could bridge the gap between image processing and medical image processing. The DICOM file is convenient because it contains most of the patient's information and the entire scan in a single file. The image data within the DICOM file is a simple 3D array. Instead of their current colour format, it would be possible for the firmware that gathers this data to save it in an array that is an equivalent of PNG or JPEG. This may increase the file size but with growing hard drive sizes this is unlikely to be a problem. Store these files in as high resolution as possible to retain all possible information. If the images are stored in a conventional image format, it would make it easier for programmers without specific knowledge of the field to contribute to the progression of medical science.

The methods used here could be altered to allow for sorting of slides from a 3D image. If slides are not in DICOM format it is possible that the slides could get mixed up. It would be possible to sort these by finding the two slides most similar to each scan and reconstructing the 3D picture.

Outside of the medical field, this program could have applications in finding the position of a frame within a video that can be viewed as a series of 2D images. A second of video can exceed 60 frames or pictures per second. Searching using an image is becoming increasingly common (for example in Google Images). Using a variation of this same approach, it would be possible to search for a specific frame in a video.

Registration between different mediums could also be a logical step forward. Because 3D scans take long to do, they are generally not something that can be done during surgery. However, some 2D scanners are

portable. For example, ultrasound devices can be made very small and cheaply. Registration between an ultrasound and a 3D scan done beforehand could give surgeons a better view of where they are in the body. This is especially useful in laparoscopic surgeries where the only visualisation is from a small inserted camera that can easily become blocked (by blood, tissue or fluid). The laparoscopic tools and surrounding tissue can be seen in an ultrasound but the scan itself is a little visually challenging. If the visible structures could be located on an MRI or CT scan, then the surgeon could see the section on the 3D scan in realtime. The part of registration not outlined here is the transformation of one image so it matches the scale and orientation of another. If this is combined with location of an ultrasound in a 3D CT scan, it could be possible to establish sizing of organs for diagnosis using just an ultrasound. In this case, one would compare a patient to an existing CT scan of a healthy person of the same sex. The software would adjust the healthy person's scan so that their proportions are the same as that of the patient being scanned. It would give rough estimates for things like kidney failure or appendicitis, both of which are deadly and diagnosable by the size of the organs. Luckily, 3D ultrasounds are already prevalent and are often used to view a baby in utero. This could be done using cheaper equipment and less processing power in low-income areas.

9 Appendix A

Code repository: https://github.com/AsthmaticDonkey/Thesis

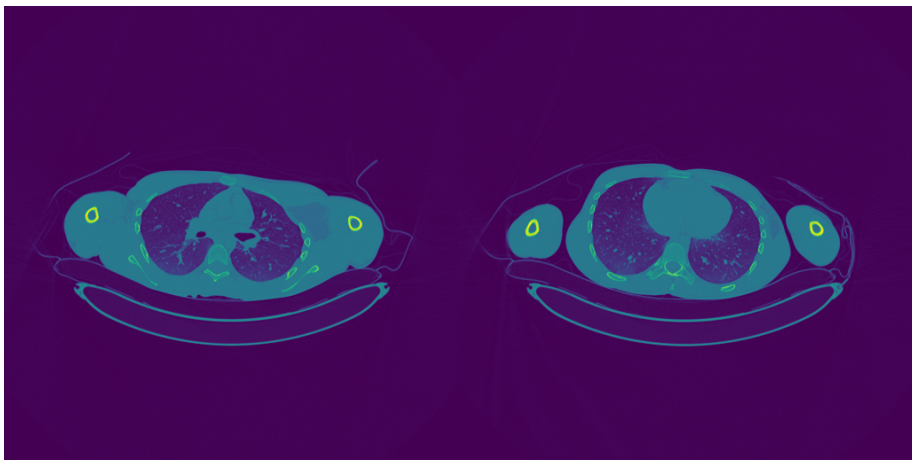

Figure 1: Scan 350 Used as a Starting Point (left) and Composite Scan at 26.57 Degrees (right)

9.1 Image-Match Dual Axes Code

The implementation of the image-match library with two search axes.

```
from google.colab import drive
drive.mount('/content/drive')
from fastai import *
from fastai.vision import *
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
from image_match.goldberg import ImageSignature
import image_match
import cv2
import nibabel as nib
import numpy as np
from matplotlib import pyplot as plt, cm
from __future__ import division
import pydicom as DCM

path = Path('/content/drive/My Drive/ThesisData')

dest = path
dest.mkdir(parents=True, exist_ok=True)

dicom_image_file_0 = DCM.read_file('/content/drive/My Drive/ThesisData/Body1.dcm')

img = dicom_image_file_0.pixel_array[350]

inti = 0
mlist = []
direction = 0

gis = ImageSignature()
```

```
a = gis.generate_signature(img)
directory = '/content/drive/My Drive/ThesisData/'

for inum in range(0, dicom_image_file_0.pixel_array.shape[0]):
  b = gis.generate_signature(dicom_image_file_0.pixel_array[inum])
  mlist.append([gis.normalized_distance(a, b),inum,direction])
  if(gis.normalized_distance(a, b) == 0):
    print('found')
    break;

tformIm = []
tformArr = []

length1 = dicom_image_file_0.pixel_array.shape[0]
length2 = dicom_image_file_0.pixel_array.shape[1]
length3 = dicom_image_file_0.pixel_array.shape[2]



for inti2 in range(0,length2):
  for inti1 in range(0,length1):
    tformIm.append(dicom_image_file_0.pixel_array[inti1][inti2])
  tformArr.append(tformIm)
  tformIm = []

inti = 0
direction = 1
for img1 in tformIm:
  b = gis.generate_signature(tformIm[inti])
  mlist.append([gis.normalized_distance(a, b),inti,direction])
  if(gis.normalized_distance(a, b) == 0):
    print('found')
  inti = inti+1


numMatch = 100
ind = 1
direc = 5

for i in mlist:
  if(i[0] < numMatch):
    numMatch = i[0]
    ind = i[1]
    direc = i[2]

print(ind)
print(numMatch)
if(numMatch>0.5):
  direc = 5;
print(direc)



if(direc == 0):
  imgplot = plt.imshow(dicom_image_file_0.pixel_array[ind])
elif(direc == 1):
```

```
  imgplot = plt.imshow(tformArr[ind])
elif(direc == 5):
  print("No Match Found")
```

9.2 Result for Single Axis Test

```
377
0.2958696000777864
0
04_76_degrees.png
368
0.2845765771032462
0
05_19_degrees.png
377
0.2925634882230837
0
05_71_degrees.png
368
0.2845765771032462
0
06_34_degrees.png
368
0.2845765771032462
0
07_13_degrees.png
368
0.2845765771032462
0
08_1_degrees.png
368
0.2845765771032462
0
09_46_degrees.png
346
0.2937259016074784
0
0_degrees.png
401
0.2866738943105918
0
```

```
401
0.2866738943105918
0
11_37_degrees.png
417
0.29391728569690223
0
14_03_degrees.png
426
0.3103164454170875
0
18_43_degrees.png
466
0.29917961619398226
0
26_57_degrees.png
588
0.30126198109746866
0
45_degrees.png
```

9.3 ORB Code from Medium.com

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Load the image
image1 = cv2.imread('./images/face1.jpeg')

# Convert the training image to RGB
training_image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

# Convert the training image to gray scale
training_gray = cv2.cvtColor(training_image, cv2.COLOR_RGB2GRAY)

# Create test image by adding Scale Invariance and Rotational Invariance
test_image = cv2.pyrDown(training_image)
test_image = cv2.pyrDown(test_image)
num_rows, num_cols = test_image.shape[:2]

rotation_matrix = cv2.getRotationMatrix2D((num_cols/2, num_rows/2), 30, 1)
test_image = cv2.warpAffine(test_image, rotation_matrix, (num_cols, num_rows))

test_gray = cv2.cvtColor(test_image, cv2.COLOR_RGB2GRAY)

# Display traning image and testing image
fx, plots = plt.subplots(1, 2, figsize=(20,10))

plots[0].set_title("Training Image")
plots[0].imshow(training_image)

plots[1].set_title("Testing Image")
plots[1].imshow(test_image)

orb = cv2.ORB_create()

train_keypoints, train_descriptor = orb.detectAndCompute(training_gray, None)
test_keypoints, test_descriptor = orb.detectAndCompute(test_gray, None)

keypoints_without_size = np.copy(training_image)
keypoints_with_size = np.copy(training_image)

cv2.drawKeypoints(training_image, train_keypoints, keypoints_without_size, color = (0, 255, 0))

cv2.drawKeypoints(training_image, train_keypoints, keypoints_with_size, flags = cv2.DRAW_MATCHES_FLA

# Display image with and without keypoints size
fx, plots = plt.subplots(1, 2, figsize=(20,10))

plots[0].set_title("Train keypoints With Size")
plots[0].imshow(keypoints_with_size, cmap='gray')

plots[1].set_title("Train keypoints Without Size")
plots[1].imshow(keypoints_without_size, cmap='gray')
```

```
# Print the number of keypoints detected in the training image
print("Number of Keypoints Detected In The Training Image: ", len(train_keypoints))

# Print the number of keypoints detected in the query image
print("Number of Keypoints Detected In The Query Image: ", len(test_keypoints))

# Create a Brute Force Matcher object.
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)

# Perform the matching between the ORB descriptors of the training image and the test image
matches = bf.match(train_descriptor, test_descriptor)

# The matches with shorter distance are the ones we want.
matches = sorted(matches, key = lambda x : x.distance)

result = cv2.drawMatches(training_image, train_keypoints, test_gray, test_keypoints, matches, test_gr

# Display the best matching points
plt.rcParams['figure.figsize'] = [14.0, 7.0]
plt.title('Best Matching Points')
plt.imshow(result)
plt.show()

# Print total number of matching points between the training and query images
print("\nNumber of Matching Keypoints Between The Training and Query Images: ", len(matches))
```

5

[5] Can be found at https://medium.com/software-incubator/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

10 Appendix B

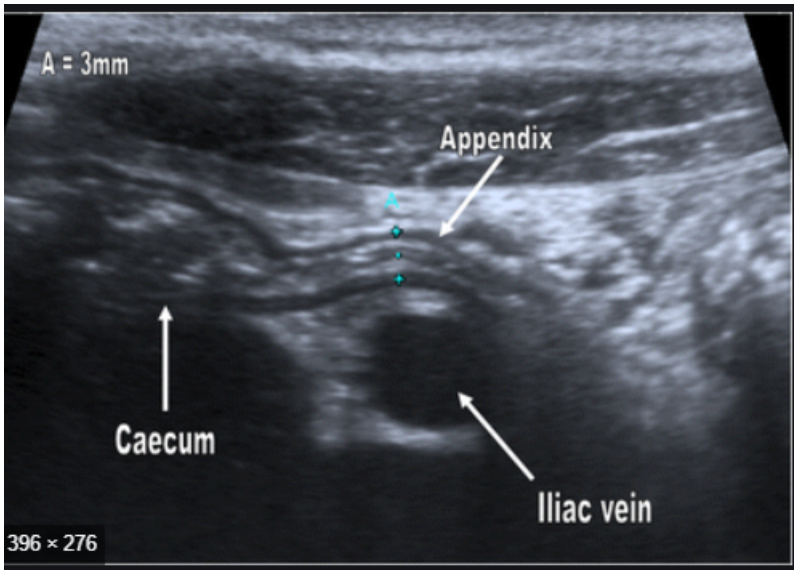

Figure 1: Appendix

References

[1] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in Medicine and Biology*, vol. 46, no. 3, pp. R1–R45, 2001.

[2] P. Lakhani, A. B. Prater, R. K. Hutson, K. P. Andriole, K. J. Dreyer, J. Morey, L. M. Prevedello, T. J. Clark, J. R. Geis, J. N. Itri, and C. M. Hawkins, "Machine learning in radiology: Applications beyond image interpretation," *Journal of the American College of Radiology : JACR.*, vol. 15, no. 2, pp. 350,359, 2018-02.

[3]

[4] D. Komura and S. Ishikawa, "Machine learning methods for histopathological image analysis," *Computational and Structural Biotechnology Journal*, vol. 16, pp. 34–42, 2018.

[5] H. Chi Wong, M. Bern, and D. Goldberg, "An image signature for any kind of image," *Proceedings. International Conference on Image Processing*, vol. 5, 2002.

[6] 2019. [Online]. Available: https://pypi.org/project/image-match/#description

[7] D. Tyagi, "Introduction to fast (features from accelerated segment test)," 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65

[8] 2019. [Online]. Available: https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

[9] 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-harris-corner-detector-32a88850b3f6

[10] R. T. Collins, "Cse486 : Lecture 06: Harris corner detector," 2019. [Online]. Available: http://www.cse.psu.edu/~rtc12/

[11] D. Tyagi, "Introduction to sift(scale invariant feature transform)," 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40

[12] Various, "Introduction to sift (scale-invariant feature transform)," 2019. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[13] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf," *Computer Vision ï£ · ECCV 2006*, pp. 404–417, 2006.

[14] D. Tyagi, "Introduction to sift(scale invariant feature transform)," 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e

[15] ——, "Introduction to brief(binary robust independent elementary features)," 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6

[16] ——, "Introduction to orb (oriented fast and rotated brief)," 2019. [Online]. Available: https://medium.com/software-incubator/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," 2011. [Online]. Available: http://www.willowgarage.com/sites/default/files/orb_final.pdf

[18] 2019. [Online]. Available: https://www.nih.gov/news-events/news-releases/nih-clinical-center-releases-dataset-32000-ct-images

[19] K. Yan, X. Wang, L. Lu, and R. M. Summers, "Deeplesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning," *Journal of Medical Imaging*, vol. 5,

no. 03, p. 1, 2018.

ETHICS APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Myrin Naidoo
Department		Electrical engineering
Preferred email address of applicant:		ndxmyr001@myuct.ac.za
If Student	Your Degree: e.g., MSc, PhD, etc.	Electrical and computer engineering
	Credit Value of Research: e.g., 60/120/180/360 etc.	40
	Name of Supervisor (if supervised):	Fred Nicolls
If this is a researchcontract, indicate the source of funding/sponsorship		
Project Title		2D to 3D medical image registration

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

APPLICATION BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Myrin Naidoo	Naidoo	24/07
SUPPORTED BY	**Full name**	**Signature**	**Date**
Supervisor (where applicable)	Fred Nicolls		24/07/2019

APPROVED BY	Full name	Signature	Date
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section 1; and for all Undergraduate research (Including Honours).			
Chair: Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the questions in Section 1.			