Automatic Alignment and Comparison of Petri Dish Images Containing Cell Colonies

by

Safar A. Alqahtani

Submitted to the Institute of Information & Mathematical Sciences
in partial fulfillment of the requirements for the degree of

Master of Information Science: Software Engineering

at

MASSEY UNIVERSITY

February 2016

Automatic Alignment and Comparison of Petri Dish Images Containing Cell Colonies

by

Safar A. Alqahtani

Submitted to the Institute of Information & Mathematical Sciences
on Feb 22, 2016, in partial fulfillment of the
requirements for the degree of
Master of Information Science: Software Engineering

Abstract

This work proposes a novel approach comprising of a chain of algorithms, for comparing, matching and aligning pairs of cell colony images taken at different stages on a Petri dish. The objective is to provide an assistive tool for microbiologists to quantify the loss or growth of cell colonies on two Petri dishes, by mapping cell colonies between a pair of images. This problem is highly non-trivial, as the shape, size and position of the corresponding colonies vary randomly. In addition, the cell colony images for comparison were taken at different times and from slightly different perspectives (i.e. effects of shearing); therefore, amplifying the complexity of the problem. Preliminary studies show that approaches purely based on SIFT or SURF, as well as algorithms used in astronomy, do not perform well on the problem domain. We therefore introduce a new approach to addressing these problems. A novel iterative technique that combines triangulation algorithms with the RANSAC alignment algorithm and AdaBoost classifier for alignment validation is proposed. Using 60 pairs of images of Petri dishes containing real biological cell colonies, we demonstrate the efficacy of the new algorithm in comparison to existing ones found in the literature. Empirical results show that the new proposed algorithm, we call K-NT for cell colonies matching, performed 4 times more accurate than other existing triangulation-based pattern matching algorithms. In the last stage of processing, we were able to generate an AdaBoost classifier with an accuracy of 98.5% that helps validate if an image was successfully aligned or not.

Thesis Supervisor: Dr. Andre L.C. Barczak
Title: Senior Lecturer

Thesis Supervisor: Dr. Napoleon H. Reyes
Title: Senior Lecturer

Acknowledgments

I would like to thank my supervisors Dr.Andre & Dr.Napoleon for their tremendous support before, during and after the period of this thesis. Also a special thanks to Dr Austen Ganley for providing the Petri dishes images and thanks to Dr Teo Susnjak for his great insight in Software Architecture . I would also like to thank the open source communities for providing a wide range of application and tools that have helped me in research and made my life much easier. These tools include, but not limited to, OpenCV, KDE, Linux, GNU, Qt, and LaTeXthe application that made this document.

Contents

1 Introduction **15**

1.1 Significance of the Research 16

1.2 Scope and Limitations 17

1.2.1 Scope 17

1.2.2 Limitations 18

2 Literature Review **19**

2.1 Introduction 19

2.2 OpenCV 19

2.2.1 Image Segmentation 20

2.3 Feature Descriptor 23

2.3.1 Scale-Invariant Feature Transform (SIFT) 23

2.3.2 Speeded Up Robust Features (SURF) 26

2.4 Matching Algorithms 28

2.4.1 Fast Approximate Nearest Neighbour (FANN) 28

2.5 Triangulation Methods 30

2.5.0.1 Groth: Pattern Matching Algorithm 31

2.5.0.2 Valdes: Pattern Matching Algorithm 33

2.6 AdaBoost 35

2.7 OpenMP 37

2.8 Qt Application Framework 38

3 Methodology **41**

3.1 Criteria for Image Selection 41

3.2 Segmentation 42

3.3 Cell Detection 44

3.4 Cell Colonies Matching 44

3.4.1 Preliminary Experiments 44

3.4.1.1 Delaunay Triangulation 45

3.4.1.2 Circle Intersection Approach 47

3.4.2 K-NT Algorithm 50

3.4.3 An Example of Cell Colonies Matching 53

3.5 Image Alignment 54

3.5.1 OpenCV 54

3.5.2 Automatic Alignment Evaluation 55

4 Results and Discussion **59**

4.1 Matching Algorithms 59

4.2 The Proposed AdaBoost Classifier 62

4.2.1 Further Optimizing the AdaBoost Results 63

4.2.2 Weak Learners 64

5 The Complete System: Software Architecture & Usage **69**

5.1 MainWindow 70

5.2 ImageHolder 70

5.3 DialogExtract 71

5.4 FiltersCV 72

5.5 DialogCompare 73

5.6 System Walk-through 73

6 Conclusion and Future Work **79**

6.0.1 Summary of Contributions 79

6.0.2 Future Work 80

A Tables **87**

B Figures **95**

C Code **99**

List of Figures

1-1 Example of matching problem. top row: Petri dish images. Bottom row: segmentation and blob counter. Note that the blobs are different in each segmentation, and that the original images are noisy. 16
1-2 All Tested Algorithms Combinations. 18

2-1 OpenCV Modules 20
2-2 Number of points to form the shape is $k = 3$ 30
2-3 Groth's algorithm. 32
2-4 Valdes Triangle (VT) 34
2-5 Triangle Space 34
2-6 Qt Framework GUI Example 39

3-1 Hough Circle Transform 42
3-2 A pair of cell colonies in Manual + SURF modes 43
3-3 Corresponding Points Example 45
3-4 Example Images for Preliminary Experiment 46
3-5 Applying Delaunay Method 46
3-6 Applying Voronoi Diagram Method 47
3-7 Applying Delaunay with Circle Intersection Method 49
3-8 Applying Voronoi Diagram with Circle Intersection Method 49
3-9 Nearest Neighbour N 50
3-10 The top 10 matching triangles using Valdes's algorithm. 53
3-11 The top 10 matching triangles using K-NT n=5. 53
3-12 Triangle Points 54

3-13 The alignment after applying RANSAC to (a) K-NT n=5 triangle and (b) Groth's matching triangles. 55
3-14 Flow chart of the current alignment rules 56

4-1 One of the pairs of images used in the experiments. Refer to Table 4.1 for SURF and SIFT settings. 60
4-2 Classification using Red & Yello Ratio Values + Green & Yellow Ratio Values 65
4-3 Classification using Red & Green & Yellow Ratio Values 66

5-1 Software Architecture Overview 69
5-2 MainWindow Class GUI: The numbers represent which class (Figure 5-1) handles the specific area. 70
5-3 ImageHolder Class GUI 70
5-4 DialogExtract Class GUI 71
5-5 FiltersCV Class GUI 72
5-6 DialogCompare Class GUI 73
5-7 The Main GUI 74
5-8 Load Image Dialog GUI 74
5-9 Image Cropping Dialog GUI 75
5-10 Filtering Toolbox GUI 75
5-11 Blobs Detection GUI 76
5-12 K-NT Matching GUI Top 76
5-13 K-NT Matching GUI Reverse 77

B-1 Image t0.100.2 96
B-2 Image t0.200.1 97
B-3 Image t1.200.3 98

List of Tables

4.1 Best parameters for SIFT and SURF. 61
4.2 SIFT & SURF performance example 61
4.3 60 pairs results 62
4.4 AdaBoost Accuracy Details 64
4.5 Confusion Matrix 64

5.1 RANSAC Matching GUI Aligned 77
5.2 RANSAC Matching GUI Misaligned 78

A.1 Triangulation Based Matching Algorithms Performance Summery . . 88
A.2 Figure B-1 Results 89
A.3 Figure B-2 Results 90
A.4 Figure B-3 Results 91
A.5 Original Attributes 92
A.6 Normalised Attributes 93

Chapter 1

Introduction

The motivation for this work came as a request for automation from the biology department: they had pairs of Petri dishes where yeast colonies could grow under certain conditions. They stamped the original set of colonies from one Petri onto a new Petri dish containing a different growth substrate to produce a similar pattern of colonies (figure 1 a) and b)). They then counted the number of colonies on each Petri to see whether there are missing colonies in the second Petri dish compared to the first.

Initially, the image processing solution for this problem looked trivial. One could use the same algorithms used to find objects in another image (e.g., a classical application of SIFT [1]), or using some of the well known algorithms used in Astronomy ([2],[3],[4], [5], [6]). These algorithms seek for a transformation matrix (scale, rotation, translation) from one image to the other. After trying both solutions, we realised that neither performed satisfactorily. SIFT [1] (as well as its variant SURF [7]) alone did not work well due to the shape of the cell colonies, since they were too similar to each other. For many of the real images we had in the dataset, there were too many false corresponding points, producing a transformation matrix that simply distorts the image without guaranteeing that the intersection between the images would be optimised. Algorithms used in astronomy rely on the fact that many stars at a certain point will match the other image. In our case, one image could have 50 blobs representing the colonies, and the other image could have several missing points. In

astronomy an additional clue can be used in the form of the brightness of stars and galaxies. In the case of the Petri dish images, such information is not always available. Despite the claim that these algorithms would only need a small percentage of the points present in both images [3], it also failed for many of the images in the Petri dish dataset. In order to make it work automatically, we needed a customised solution for the specific images that were being assessed. This work describes the approach taken to produce a more acceptable solution for this particular problem.

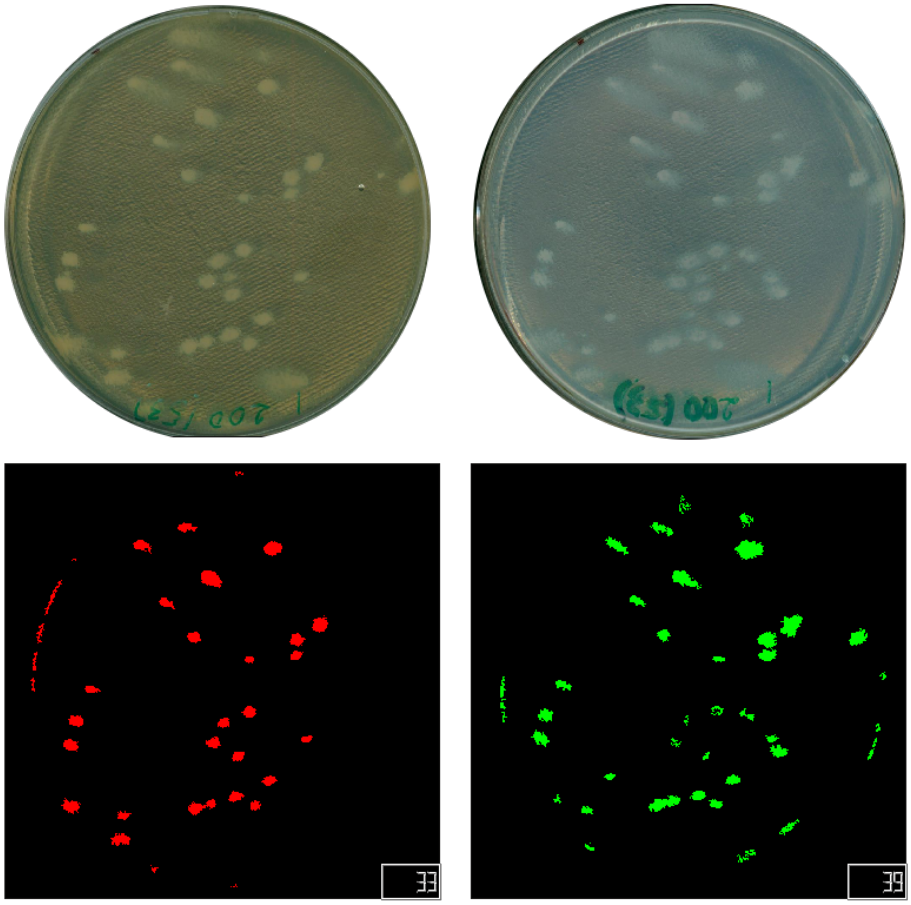


Figure 1-1: Example of matching problem. top row: Petri dish images. Bottom row: segmentation and blob counter. Note that the blobs are different in each segmentation, and that the original images are noisy.

1.1 Significance of the Research

Microbiological research typically involves monitoring and comparing the growth and demise of cell colonies in petri dishes, in the thousands. This research mainly con-

tributes in the automatic matching and alignment of colonies between a pair of petri dish images taken at different stages. The complete system requires a chain of algorithms working in succession, starting with cell colony matching, then alignment algorithms, and lastly an algorithm for validating the results. We propose an algorithm, called K-NT to solve the problem of matching colonies more accurately. K-NT can perform up to 4 times better than existing triangulation-based, namely Valdes [3] and Groth [2] algorithms. K-NT, in combination with the RANSAC algorithm for cell colony alignment produces results with an increased accuracy of 51.6% compared to Valdes's with RANSAC and 65% compared to Groth's with RANSAC. AdaBoost classifier was used as the results validator, and was able to achieve the accuracy of 98.5% when classifying the correctness of an alignment image.

The preliminary results of this research were peer-reviewed and published at IVCNZ 2015 [8].

1.2 Scope and Limitations

1.2.1 Scope

The proposed algorithm K-NT has been tested on a real set of biology data, these data are a collection of more than a hundred images and each image holds 2 to 6 Petri dishes samples.

In Figure 1-2, the squares represent the feature extracting algorithms, the triangles represent the matching algorithms, and the circles represent the alignment algorithms that were used in this research.

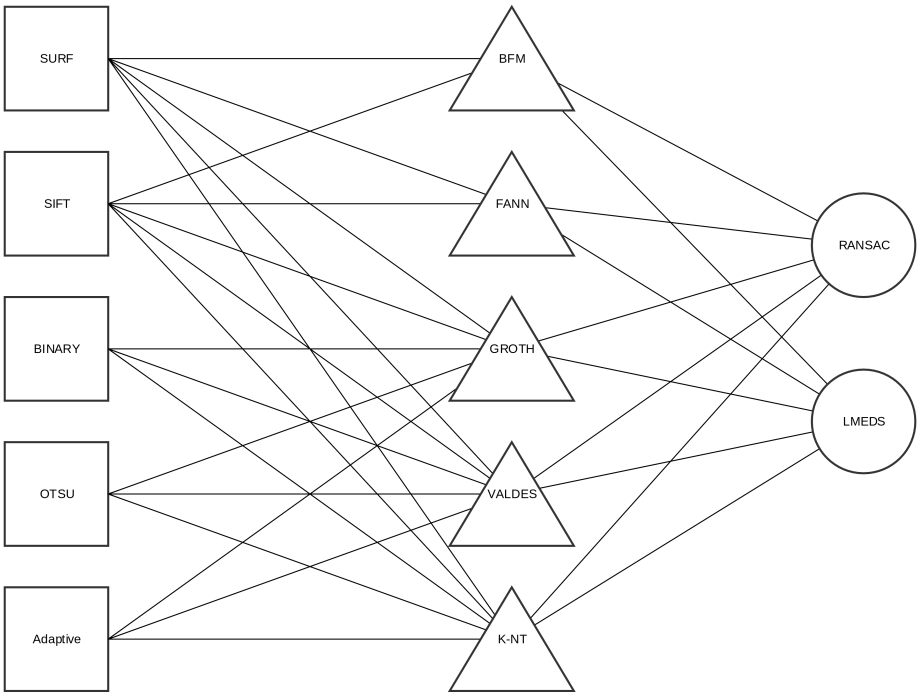


Figure 1-2: All Tested Algorithms Combinations.

1.2.2 Limitations

The Limitations of the proposed algorithm and classifier are as follows:

- K-NT is similar to other matching algorithms which will take a very long time to process if more than the recommended cells count where used.
- K-NT is designed to work with cells positions only and will require further modification if other inputs were considers such as, brightness, size, etc.
- The proposed AdaBoost classifier will only work with the features extracted during the alignment stage.
- Blurry images that shift some cells positions might reduce the accuracy of both the K-NT and the AdaBoost classifier.

Chapter 2

Literature Review

2.1 Introduction

Replica plating [9] is a microbiologist method in which the master Petri dish is reproduced by pressing it against a special covered disk, this process is not perfect as the resulted dish will never be a real replica. One purpose of this technique is to enable microbiologist to do a wide range of experiments on the replica instead of ruining the master dish. After experimenting on the secondary dish microbiologist will compare the secondary dish with the original one to see if the colony is sensitive to a specific substance.

2.2 OpenCV

Processing images is nothing new to computer science. In 1959, Image Processing was used to enhance the quality of the moon images taken by NASA [10], OpenCV is an open source library that was established by Intel to process images in year 2000 [11]. The goal of OpenCV library was to simplify image processing where the developer does not need to reinvent the wheel [12].

OpenCV offers a great collection of models each provide a set of useful shortcuts. In the Figure 2-1 below is a description of the modules offered by OpenCV which was inspired from the book "Instant OpenCV Starter" [13].

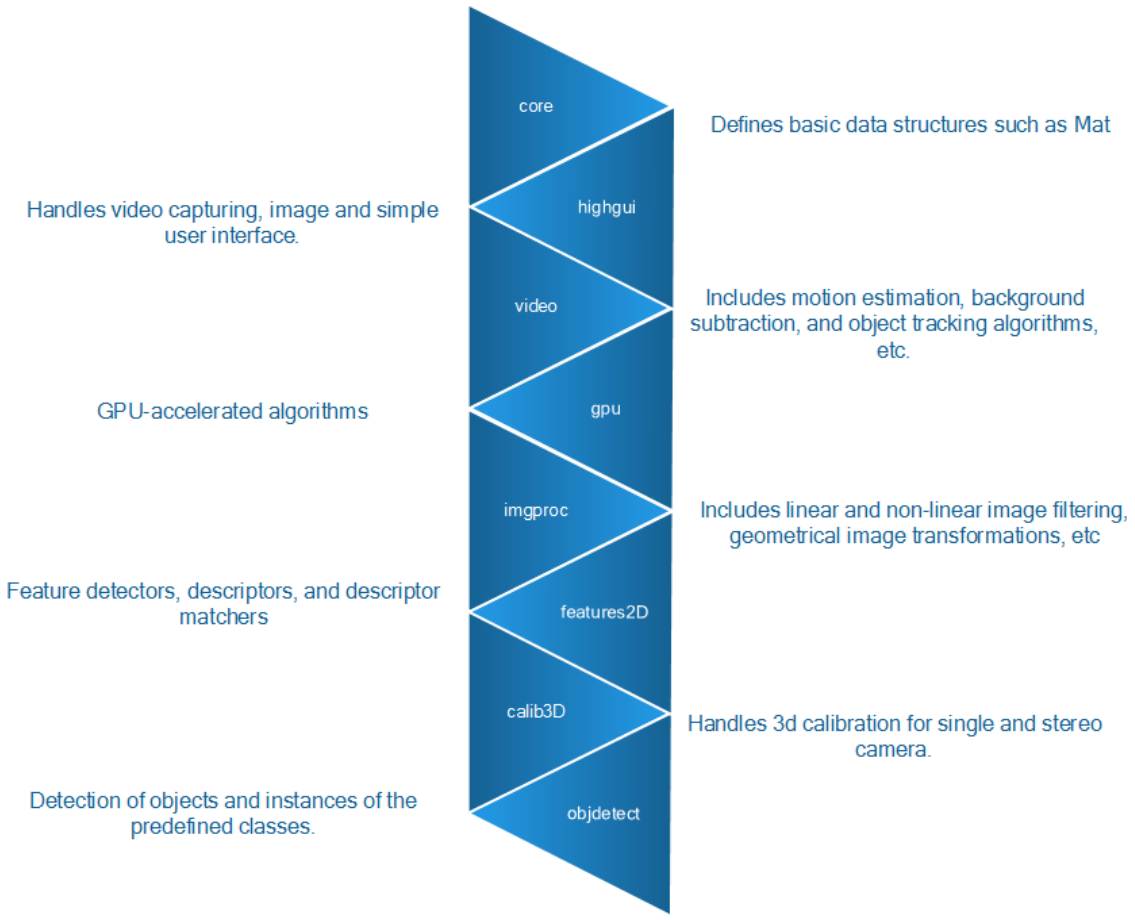


Figure 2-1: OpenCV Modules

Popular applications that uses OpenCV [14]:

- Street View Panorama
- Vision system of the PR@ robot
- Robots for Mars exploration
- Quality control of the production of coins

2.2.1 Image Segmentation

Image segmentation is a term related to computer vision and complex algorithms which have proven to be very useful in the contemporary era. The term refers to a process which breaks up a single image into many different images. This breaking up

is known as segmentation while the multiple images that the database produces, are the segments. The idea is to separate and then gather pixels such that they become representation of parts of image regions [15]. Once the segmentation is completed, there are numerous ways in which the image can be used. Since the field has so many potential applications that can really make a difference in the way we look at things programmers are constantly looking into this field. Hence there is plenty of literature available about the uses that this algorithm can serve and how exactly does the dynamics of such a process work. In their paper, Krishna Kant Singh and Akansha Singh [16] explained in immaculate details, exactly how the process works and the types of segments that can be made.

According to their research, the ultimate purpose of all image processing endeavors by programmers is to recognize the objects of interest in a better and quicker way than one can by looking at it through naked eye. The algorithm works by separating pixels, by defining them by being either if interest, or of no interest [16]. Through this simple process, a binary image is produced, which as the name refers, is coded in binary language. If the pixel belongs to the object, it is given the value one; otherwise the value allotted to it is zero. Once the binary coding is completed, the images are segmented. There are four methods which are adopted for this process. They include Threshold method, Edge based method, Region based method and connectivity-preserving (model) based method [17]. According to the thesis of Yatharth Saraf [17], the threshold method is used when the object is clear and intensely visible against the background. The edge based method operates with the identification of the edges of the image, but is again not effective if the images in question are blurry. The Region based method makes use of the intensity in the images, and segments the object with respect to their intensity and clarity in them. This approach will over merge the blurred regions and give a viable result. The last method, the one of connectivity-preserving, initiates the forming of complete boundaries and modifies the results with respect to energies and their elasticitys. Using the threshold segmentation technique in particular, precise image edges can be produced for estimation of exact thresholds within an image. It separates the variance of the

grayscale values such that the edges of the image are found and hence it becomes rather easy to separate the object from the background of the image [18]. In order to use the newly segmented images for various purposes, the segments need to be compared with each other. On the basis of the comparisons which are drawn and the results which are concluded, the segmented images are used for their respective applications. A research paper produced by Caroline Pantofaru and Martial Hebert [19] laid down the framework which is to be used in the comparison of segments. For such a comparison, three characteristics are to be looked after, which include the correctness of an image, the stability within an image with respect to the choice of the parameter which is adopted and lastly the stability within the image itself [19].

Researchers have also looked into the performance of different algorithms of image segmentation through various tests, to look at the responsiveness, accuracy and speeds of different methods [20]. There are numerous applications of image processing algorithms, once the images have been successfully processed. In a paper submitted to Department of Electrical Engineering of a Belgian University [21]; the possible clinical uses of image segmentation have been laid down. The most amazing use of segmentation is the assigning of pixels into 2D and 3D images, which render them useful in the field of radiology in particular [22]. These 3-dimensional visuals enable the model to be used in the processing of data related to images which makes analysis of the internal structures rather easy. Today, this technology is being used for medical purposes and has become a dominant part of our hospitals and clinics. These parts have become so important that the treatment of today is more or less impossible without them. From the ultrasounds to X-Rays, the image segmenting elements in the radiology play a very major role in pre diagnosis tests, the actual operations and the post operation feed back as well. When these images found through algorithm based segmentation are combined with human expertise, any inaccuracies in the computer based process are eradicated. Other uses of segmentation include object recognition in a given background. This is what has made the cameras of today advanced enough to recognize faces and have cool features like smile shutters. It is used in looking up for a given image through a data base of images, as the images with similar segments

will be recognized and compared. The results which end up being alike are separated. This makes it a useful security tool in surveillance cameras, to locate criminals on the run through CCTV cameras in different streets.

Besides these huge features that have broken significant ground in the human life and contributed in huge ways to make our generation take a leap towards becoming the advanced one that it is today, there are other little applications of this tool as will. These include things like photo editing, image compression, finger print scanning, brake light detection, iris recognition, etc. The potential that segmentation holds is limitless and this is the dominant reason why further effort and research is directed towards this field.

2.3 Feature Descriptor

2.3.1 Scale-Invariant Feature Transform (SIFT)

The study relating to image processing and image matching algorithms is an extensive one. With an array of highly developed features available at hand, the process of selecting a particular one has become rather difficult. To find the best fitting feature to serve any given purpose, a comparison is drawn over their performance and robust nature in particular. Making the comparisons among the available options easy is the extensive amount of literature that is at hand over the web. The literature not only defines SIFT and all the dynamics of the feature, but at the same time it jots down the numerous applications that are affiliated to it and the draw backs that exist within the model.

Published by David Lowe in 1999 [23] and currently patented in US by the British Colombia University, the Scale-invariant feature transform (SIFT) is a computer vision based algorithm. The features if used for image recognition it is required to be unaffected by some core changes in the dominant parts of it. First and foremost, it needs to be highly specific and efficient in its identification of particular image from a given set of images. To render its purpose of creation, the feature needs to work

regardless of differences in the illumination among objects, the common variations among objects (like rotation, scaling, transformation) and invariant to the 3D projection transformations that are generally found [24]. Where the previous features were highly sensitive to any changes that were administered in illumination and projective distortion, SIFT has found a way through these changes to give results that are dependable and more accurate than those acquired through the earlier versions. What makes this technique so good is the fact that it initially transforms the image into a vast collection of vector features that are local. In another paper published by David Lowe in 2004 [25], the feature and the method to make use of it is elaborated in further detail.

There are four major stages involved in the computation and generation of image features. The first stage is that of detection. In this stage, the system scans for scales and locations within images for interest points. The ones which are invariant to the changes in orientations, scales and illumination are identified using Gaussian function [25]. In the second stage, key points are identified on the basis of the scale and location. The selection of these points is premised on stability measures. The third stage is that of orientation where orientations are allotted to key points based on the gradient directions of local images. The comparisons of all images are then drawn on basis of the invariance found within the features marked by different orientations. The final stage is that of key point description, where the gradients are compared with respect to regions around each key point. This is where the representation allows for significant distortion in shape as well as illumination, and draws a match based on the key points, regardless of these distorted differences.

Further research elaborates what makes this detector of SIFT so accurate. Firstly, it discredits the responses that hold very low contrast [26]. Hence the images that are distorted by noise and cracks end up being eliminated if they fall above or below the peaks of the threshold. This removes element of doubt from the results procured. The doubt is further eradicated when the key points having unstable and edgy responses are also eliminated. This gives a higher degree of accuracy to the matches that are found. There are numerous applications of this feature, with particular trend growths

seen in recent years.

Researchers have made use of this technique in iris recognizing softwares to grant authorization [27]. Since it makes use of similar images from a given database, eradicating any element of doubt caused by low contrast or shaky features, it works best for iris scanning in high security zones. The paper [27] presents a procedure of adopting SIFT for iris recognition in biometric authentication, making the process even more authentic than ever. It proposes a three tier format for the scan, beginning with segmentation of the new images that are gained. The next step is matching those images with the database which has already been stored. The last step is the evaluation, where the system decides if the match is legit or should it be discarded for failing to meet the pre-set criteria. It will be more effective, since this method works irrespective of the scale, rotation and even illumination to some extent. Not only is this method a lot faster, but more accurate as well in catering to its application of providing fool proof security. Although this method has spread really fast and is used in many applications owing to its excellent performance and high level of accuracy, the method has its draw backs. The applications that run on SIFT require a lot of time and energy beforehand. Not only are they time consuming, but also happen to be highly complicated [24]. Instead of rendering SIFT useless acceleration algorithms like Compute Unified Device Architect (CUDA) have been developed. The use of these accelerators reduces the response time and makes the processes making use of SIFT a lot faster. It works on the premise of a linear relation with the number of key points held in SIFT.

There are many places where SIFT has potential to grow further and evolve into an even better feature than it already is. From the currently operational iris scanning, the system can take up to the larger face scanning as well, which will work exceptionally well with its performance in effective recognition methods. It can also give room to feature based model, and break significant ground in 3D pictures and model making [28].

2.3.2 Speeded Up Robust Features (SURF)

Speeded up robust feature, abbreviated as SURF, is a revolutionary feature in the detection and description of objects. Introduced in 2006 [29], it has significantly boosted up the technological ascend of mankind by making astonishing use of algorithms. The task of finding correspondence between given visual elements has always been an ultimate target that the tech experts have been striving for. Numerous computer applications and softwares have been designated to pointing out the unique similarity that unifies the images in question. Not only does it aid in object registration, object recognition and object classification but it has also been used in 3D reconstruction recently. Compared to other features serving the same purpose like SIFT, SURF is a lot more robust. The detection of discrete image points is a complex process, which is interrelated into 3 tiers [7].

The first step is of detection of interest points. In this stage, the system automatically detects and identifies features of interest. These are usually at specific locations within an image, for instance at the T-junctions, corners and blobs. The images are repeatedly seen through the same distinctive locations. The features that are detected over and over again, regardless of changes in the view points are the interest points of the given images. This detection make use of the concepts that are typically applied within SIFT algorithm as well, but the results acquired in this way are a lot more credible, dependable and quicker to achieve. Once the detection is completed, the second stage is initiated, which is known as the description stage. This is required to provide a robust and tangible input as to the similarity of the interest points. It ensures that the features work regardless of the scale and rotation involved. Instead of concentrating on the feature of the image itself, the descriptor takes a different approach. It makes use of the surroundings of the point of interest, by looking into similarity of the objects existing around that particular point. As a result, integral images are created by making use of the Haar wavelet responses. The area of interest (which is the area surrounding point of interest) is rotated in various ways for this descriptor stage to be effective. Hence it becomes prone to changes in

noise and formation of the point and its surroundings. The last stage is the labeled as the matching stage. This is basically the process where the resulting images from the descriptor stage are matched. This matching is initiated by quantifying the vector distances and the time that is taken for those distances to be covered. In this way, the common features between different images that are scaled and cropped can be identified. Putting this technique into perspective, location of people, objects and faces can be easily initiated. In the modern world, the security agencies and corporation make use of SURF for the speedy tracking of objects and people for investigation related purposes. At the same time, this feature is used in making 3D scenes as well. As stated by another research carried out by Jacob Toft Pedorsen, SURF analysis and detects the interest points by making use of the Hessian matrix. This matrix is used primarily because of its accuracy and efficiency [30]. The scale selection is also done through the arithmetical approach adopted by Hessian. Although this technique involves cropping, which leads to a reduced number of results as a significant loss in repeatability is observed yet the high speed and excellent performance of repeaters in this method outweigh the loss of images.

According to the research conducted and presented by Pinto & Anurenjan [31] in a conference, SURF is an important tool which can be used in video stabilization. Using this feature, the videos that have already been made can be edited in professional manner. Their research tells how a given video can be enhanced and altered, with the removal of motion from places, while the addition of further motion in other places. In order to maintain a video which is stable and properly sequenced, a motion separation can be used to establish movements that are unnecessary and not needed while global camera motion estimation helps with the flow of events. Using SURF, video can be broken into several frames [31]. This frame by frame division of the recorded version not only allows for filtering and smoothing of motion vectors, but also enables for appropriate point trajectory features. The frame by frame stabilization is used not only in video editing, but also in reviewing the minor details in given videos, particular the ones obtained from security cameras. In a similar conference, a group of researchers presented a concept of detecting copied videos using matching

phases and finger print extraction of SURF [32]. They presented and published their findings, which encompassed details regarding the use of frame by frame sequencing and matching phases to jot down similarities in video content. This technique is used by major video based websites to find the copied content in their databases, without going through the trouble of manually looking into every single video available. Perhaps one of the most significant applications of SURF in the contemporary era is on the military fronts. Using SURF, unmanned aerial vehicle (UAV) industry has found a breakthrough [33]. Not only has the cost reduced now with sensors that are cheaper to install, but the accuracy of the destination of sensors has also been enhanced. Research has been conducted to evaluate the performance of SURF using various dimensions within the descriptor. In the same research, the sample points have been changed so that the matching accuracy was altered just to test the performance of SURF. The variations in the dimensions and point of focus have yielded an array of results, which concluded that not only is SURF effectively efficient, but has broken significant ground in improving the UAV in military operations. This technological leap has changed the warfare techniques and the ground battles as history has been recording them.

2.4 Matching Algorithms

2.4.1 Fast Approximate Nearest Neighbour (FANN)

Artificial intelligence has helped humanity leap forward into the highly advanced generation that it has become today. With the systems becoming self sustaining, the demand for advancing technology is further increasing. Fast Approximate Nearest Neighbor (FANN) is a procedure which makes use of procedures that are efficient on a computational level, to identify objects that are similar with the objects pre-provided in a huge dataset [34]. The analysis of similar videos from the video based content is a very lengthy and hectic process. It is not possible to keep track of video feed for the purpose of surveillance. Reopening and re watching all of the videos will

be a task next to impossible. Through Fast Approximate Nearest Neighbor, a very quick analysis from those videos can be drawn and the different one can be located without the lengthy process of going through all of them [35].

There are two methods that are required to be seen into before the complex features of nearest neighbor can be applied. They include spectral hashing and non-Euclidean spectral hashing. According to a paper produced by Esmaeili, Ward and Fatourechi [36], FANN has been proposed for use in hamming space as well. The proposed feature of FANN is called Error Weighted Hashing (EWH), working on an algorithm which performs 20 times faster than the previously used one. Not only can it be used in detection of plagiarized videos, but at the same time it can be used in retrieval of media as well as binary fingerprinting. Not only is it a lot faster, but the results yielded through this process are way more accurate as well [36]. When it is about the algorithms based on high-dimensions, no exact ones have been produced so far. The ones that are available are conflicted and involve a lack of accuracy. But when compared to the linear search methods, the latest ones are a lot faster. The speed difference is such that there comes a time when the difference of accuracy becomes minimal and ignorable [37]. Hence it is ignored, and one of the methods of Fast Approximate Nearest Neighbor is adopted to speed up the process.

According to research conducted in Columbia university computer science department, the best and fastest performing algorithm is k-d tree. This conclusion has been drawn after extensive research, yet the best algorithm finally depends on the given dataset and the degree of precision desired by the user. These methods of artificial intelligence not only make the process of selection easy, but reduce the load of work which is being levied on the manual labor. Where it speeds up the video based analysis, the process makes automated configuration a task which is easily catered. There are several implications of this process, based on numerous amounts of research which has been conducted, not only with respect to artificial intelligence alone, but the entire process of FANN in general.

2.5 Triangulation Methods

The most popular methods to match 2D points use some form of triangulation. A simple matching approach uses the triangles formed by different objects (in astronomy this could be stars, in our case blobs of cell colonies). The vertices of the triangle are the centre of mass of each object. In this case, there are many possible combinations of points. The number of triangles can be estimated using the Permutation equation:

$$P_n = \frac{n!}{(n-k)!k!} \tag{2.1}$$

Where

$$\begin{aligned} n &= \text{Number of points in the list} \\ k &= \text{Number of points to form the shape} \end{aligned}$$

Therefore the number of triangles is proportional to n^3. This limits the number of objects being used in the match, otherwise it quickly becomes computationally intense (both in memory and CPU cycles).

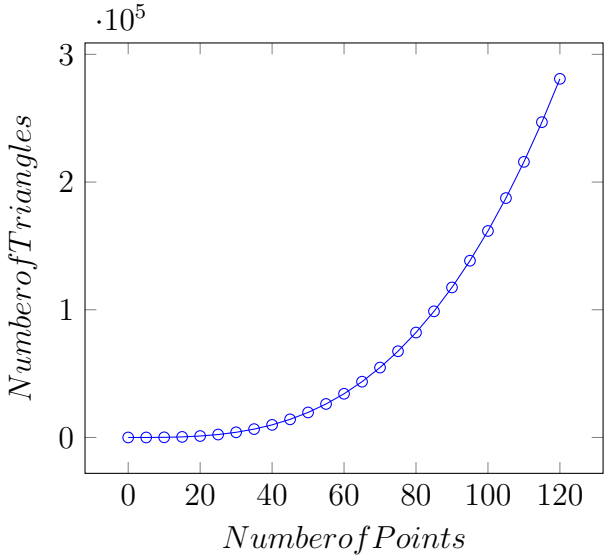


Figure 2-2: Number of points to form the shape is $k = 3$

2.5.0.1 Groth: Pattern Matching Algorithm

In 1986, Edward J. Groth invented a pattern matching algorithm which was named Groth Algorithm. Groth [2] designed his version of triangulation to solve the classical astronomical problem of matching pairs of stellar images taken from different view points or different cameras. Groth applied the algorithm to astronomy problems, but other applications such as in biology have also benefited. For example, Groth's algorithm was used for the identification of whale sharks ([38]). Both problems rely in that the majority of the triangulation points (stars or blobs in the whale shark skin) do not radically change over time.

The algorithm assumes that you have two two-dimensional lists of points, each point contains a normalized values of x and y in each dimension. Before going to the next step Groth recommends that the points (n) in each list should be decreased to the lowest possible since the algorithm requires a computation time up to a high power of n. In his case the stars have different brightness so a filter can be applied to select the brightest stars only. To filter the list even further, any two points in one of lists are closer than E which represents a user defined variable should be eliminated from the list to avoid a potential mismatching in the other list.

The algorithm starts by selecting a two-dimensional list of points, each point containing normalized values for the Cartesian tuples. The number of points should then be reduced since the number of combinations grow very fast and may render the computation infeasible, even for a few hundred blobs. To achieve this reduction, a minimum distance parameter is adopted. After the points have been stored, a list of all combinations 3 by 3 (all triangles 2.1) is computed for both lists of points.

The algorithm stores each triangle based on the side length between each two points, the longest side should lay between points 1 and 3, the shortest side should lay between points 1 and 2, and the intermediate side should lay between points 2 and 3 as shown in figure 2-3. By sorting the three sides of the triangle the ratio of the longest side to the shortest side is then calculated and stored. The angle S in figure 2-3 is also stored for each triangle. Triangles are then matched according to

their parameters (tolerances in equations 2.5 and 2.7). The best matches are stored and sorted by similarity.

At this stage many false matches would have been computed. Each matched triangle cast three 'votes' for each one of its vertices. The points that get more 'votes' have a better chance of being a true match. The voting data is sorted so that the points with more 'votes' are chosen, until one of these three conditions happen: either the number of votes (for a specific point being chosen) drops by half in relation to the largest voted point, or a point that was already assigned gets assigned again, or the vote is zero. The result is a selected number of points with possible matches between the two images.

To protect against false matches, the entire algorithm can run again in exactly the same way, but using only the points that were chosen in the first run. If all points match again, this is an indication of a good match. If not, after a few iterations the number of points can get very low (or zero), indicating that there is no match between the images.

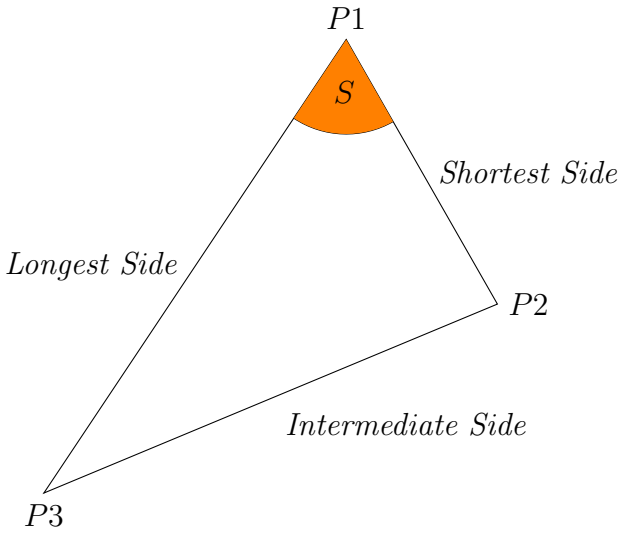


Figure 2-3: Groth's algorithm.

For completion, we repeat the formulas found in [2]. For each vertex in P_1, P_2

and P_3, the ratio:

$$R = \frac{r_2}{r_3} \tag{2.2}$$

Where:

$$r_3 = \sqrt{\Delta x_3^2 + \Delta y_3^2} \tag{2.3}$$

$$r_2 = \sqrt{\Delta x_2^2 + \Delta y_2^2} \tag{2.4}$$

and the tolerance in R is

$$t_R^2 = 2R^2E^2(\frac{1}{r_3^2} - \frac{C}{r_3 r_2} + \frac{1}{r_2^2}) \tag{2.5}$$

where C is the cosine of the angle at point one. C and its tolerance are given by

$$C = \frac{\Delta x_3 \Delta x_2 + \Delta y_3 \Delta y_2}{r_3 r_2} \tag{2.6}$$

$$t_c^2 = 2S^2E^2(\frac{1}{r_3^2} - \frac{C}{r_3 r_2} + \frac{1}{r_2^2}) + 3C^2E^4(\frac{1}{r_3^2} - \frac{C}{r_3 r_2} + \frac{1}{r_2^2})^2 \tag{2.7}$$

where S is the sine of the angle at point one (see figure 2-3).

2.5.0.2 Valdes: Pattern Matching Algorithm

Valdes's algorithm [3] is also based on triangulation and it is a variation of Groth's algorithm. Valdes's approach unlike Groth's does not eliminate points in a list if they are too close to each other since his algorithm can endure some misidentification and close objects can produce reasonable coordinates in the triangle space. Valdes's triangle generation is based on Stetson's method [39], transferring a triangle's sides into a single point in a 2d triangle space defined as:

$$x_t = \frac{b}{a}, \quad y_t = \frac{c}{a} \tag{2.8}$$

The following triangle in figure 2-4 can be represented in the triangle space in figure 2-5, if we took side a,b,c lengths in a decreasing order and computed them using the equation 2.8.

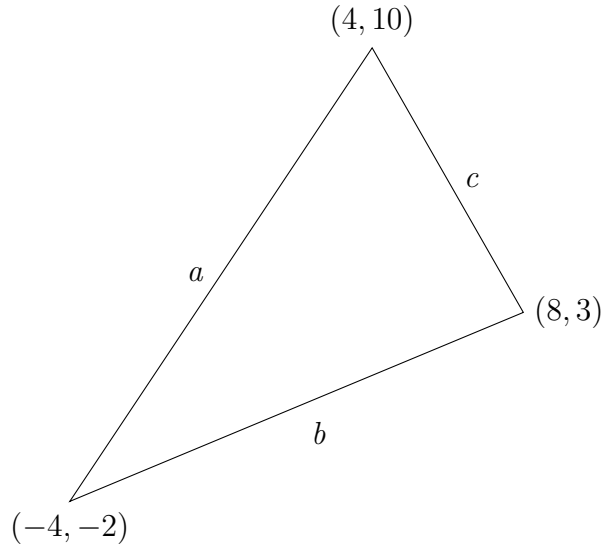


Figure 2-4: Valdes Triangle (VT)

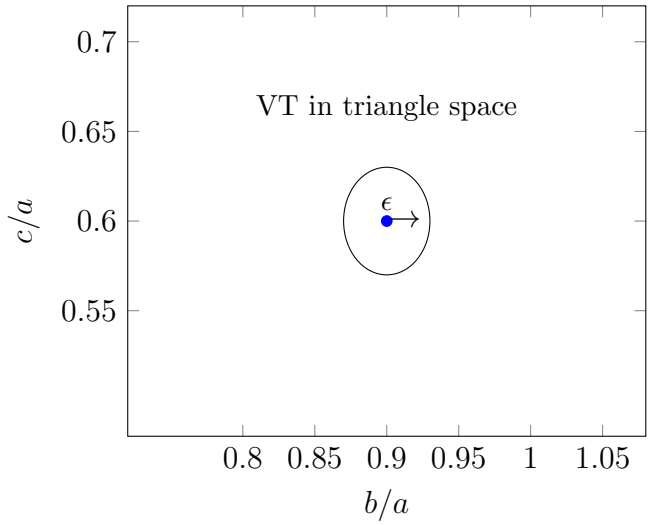


Figure 2-5: Triangle Space

The end result should be two lists of points that are a representation of all possible triangles in the triangle space. Points that are close to each other in the two lists should be considered as a possible match independent of offset, inversion, rotation and scale between the original images.

Triangles are matched when their coordinate in the triangle space is closer than a specified distance ϵ shown in figure 2-5. The simplest way of finding a match would be looping through the two lists and computing the Euclidean distance between two points (one from each list or image). This method is slow and can be easily optimized by using a binary search instead.

2.6 AdaBoost

AdaBoost is among one of the most extensively studied and used algorithms, known widely with its shorter name. Short for Adaptive Boosting, the algorithms deals with the machine learning mechanisms. Instead of involving single inaccurate procedures like the rule of thumb, which are little more than guess work, AdaBoost maintains concrete machine learning using a combination of various rules that yield inaccurate and weak results.

Consequently, the AdaBoost algorithm becomes the most accurate prediction rule. It was designed by the combined efforts of Robert Schapire and Yoav Freund, who have been appreciated and given awards for their amazing work. Since it is such a ground breaking innovation, extensive amount of literature has been produced on it. Robert E. Schapire, one of the creators of the algorithm and a professor at Princeton University explains the perspectives of AdaBoost. In his paper, he discusses how it is a learning system, and makes use of the concept of eliminating the odd one out. It does not make use of a new, highly evolved technique. Instead, it takes the seasoned and old techniques which are highly inaccurate, and makes use of them to produce precise results. This is achieved in a very smart process, where the inaccurate process are applied multiple times such that outlier values are eliminated, leaving the highly accurate and correct solutions [40]. In his paper submitted to MIT (Massachusetts institute of Technology), Tommi Jaakkola writes about the practical advantages of using AdaBoost. He mentions that it is a very fast system, which happens to be rather easy to use as well, owing to its simplicity [41].

Among the best features of this algorithm is its flexibility, which enables it to

be used with a combination of programs. Hence it is a highly versatile program, which can serve multiple purposes. Even the data that it operates with needs not be restricted as it can comprehend textual, discrete, numeric, binary and more or less any other type of data set. From programming to predictions and detection of similarities, the uses of AdaBoost are limitless. In their research paper, students of Colombia University presented a list of applications among which AdaBoost can be distributed. Their proposal was to use this mechanism for classification of datasets that vary highly and are distributed such that fitting them into memory is a nearly impossible task. The study proposes the use of this algorithm such that it helps in on-line, scalable and distributed learning [42]. This will enable it to be used in application systems like JAVA and JAM. There are other places where this system can be of critical importance and extreme help. Research has been conducted to put things into perspective and try to make use of this algorithm in basketball games to identify players [43]. After conducting a series of tests on simple moving objects instead of the entire games with the same backgrounds, it was found out that the algorithm is sensitive to the noise in the background. This noise is the reason which ended up distorting the results to such a degree that the detection of players was found out to be accurate only 70.5% of the times. The accuracy may seem impressive when compared to other methods, but the fact remains that it is not nearly close to the amount required for recognition of players in actual games. Where AdaBoost will be rendered useless in recognizing the players through the complete body parts, the same program is found out to be very effective in recognizing the body parts on their own. This indicates that the algorithm holds potential in recognition of singular parts, like facial recognition. It may be inaccurate in real situations, particularly in sports related events, but its fast pace of scrutinizing through the immense quantity of data and ability to recognize features as long as they do not involve multiple aspects, make it a very suitable system for object detection.

In addition to the detection of objects, the algorithm works perfectly for image restoration, that as well without any prior knowledge at hand [44]. Through this method, both the distorted and degraded images can be perfectly restored. According

to a paper submitted to the University of Los Angeles, by X. Chen and A.L. Yuille AdaBoost can be very useful in detecting and reading texts within the scenes of a city. This has lead to a very useful application go the algorithm, which had provided ground breaking assistance to the people who are visually impaired. Not only has the algorithm given very speedy responses (as fast as processing of images of size 2048 by 1536 within 9 seconds), but at the same time it has also given high quality results which are accurate up to 93% [45]. But of all its applications, perhaps the most significant one is related to the assistance that AdaBoost has provided to the systems of face detection [46]. Using an array of complex design techniques like those of classification, image scaling, pipelined processing and integral image generation not only is the process of face detection possible, but is also accelerated to an incredible speed. This process makes use of Integral images and Haar features, while ignoring things like size, condition and position of the face.

The ultimate product is used for many purposes in different places in the contemporary era. It is used for security purposes in computer interfaces and smart rooms, for granting access to some specific people, while denying it to others. The intelligent robots of today also make use of this technology to integrate and work with humans in controlled environments. It has rendered its use in image analysis of biomedical degree. From surveillance cameras to tracking criminals through CCTV footage, the incredible amount of security related advantages that we claim and make us of today are a product of AdaBoost algorithm.

2.7 OpenMP

Optimizing algorithms without making them complex can be a hard thing to achieve, running slow algorithm can be time consuming and might slow down productivity. OpenMP is one of the most known APIs for parallel computing. OpenMP API is available on Unix and Windows platform which make use of the Shared Memory Model. OpenMP API can be called from C, C++, and Fortran programming languages. In this research OpenMP was used to parallelize for-loops, achieving a parallel loop was

made easy by OpenMP. By adding "#pragma omp parallel for" before the for-loop it will parallelize it.

Listing 2.1: OpenMP Code Example

```
std::vector<Triangle> a_triangles = getSetOne();
std::vector<Triangle> b_triangles = getSetTwo();
#pragma omp parallel for
for(size_t i = 0; i < a_triangles.size(); ++i){
    for(size_t j = 0; j < b_triangles.size(); ++j){
        double distance = DistancePoint2d(a_triangles[i],
                                          b_triangles[j]);
        if(distance < a_triangles[i].distance){
            a_triangles[i].distance = distance;
            a_triangles[i].match = &b_triangles[j];
        }
    }
}
```

In Listing 2.1 the code is being optimized using OpenMP library, by using line three we have enabled the first "for loop" to be parallel which means each available thread will be assigned to handle one iteration of the first loop. By using one line of code we are able to achieve better overall speed and will preserve the simplicity of the algorithm.

2.8 Qt Application Framework

OpenCV is a great library for image processing, but as expected they do not provide a solid User Interface (UI) instead they provide a simple function that will show a window with the selected image. OpenCV window can only have one image in it and any number of customizeable trackbar. These UI elements are provided by OpenCV for testing purposes.

Qt is a cross-platform application framework that is used for UI and application development. Qt comprises the Qt framework with modular cross-platform C++ class Qt libraries and Qt development tools including an integrated development environment, Qt Creator IDE, and productivity tools. Using Qt GUI as a frontend for OpenCV library will make the application more stable, customizable, and user friendly.

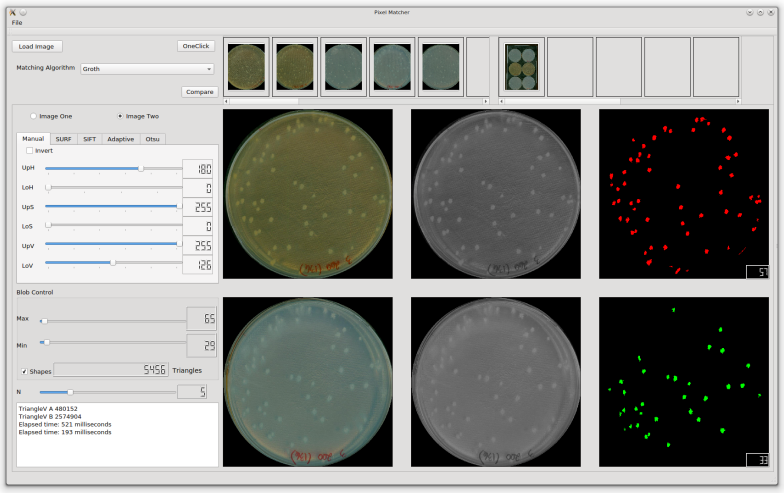


Figure 2-6: Qt Framework GUI Example

In this research we have used Qt framework to create a user friendly GUI as shown in Figure 2-6, from this GUI the user will be able to do everything without being involved in the code.

Chapter 3

Methodology

In this chapter we are describing all the needed steps to tackle the Petri dish problem that includes, how to select a proper image of Petri dish, how can image segmentation help in refining the image, how cells are being detected, how two sets of points are matched, how to align two sets of points.

3.1 Criteria for Image Selection

INPUT: Image that contains Petri dishes
OUTPUT: Two cropped Petri dishes

The proposed algorithm can work with any image that has consistent pattern which can be extracted. To elaborate more, a consistent pattern means a set of points that are placed at a fixed location from each other. For the algorithm to function properly an image with more than three features should be selected. The application was mainly designed to to automate the Petri dish detection, so the OpenCV's implementation of Hough Circle Transform [47] was used as an option to automate the location of a Petri dishes.

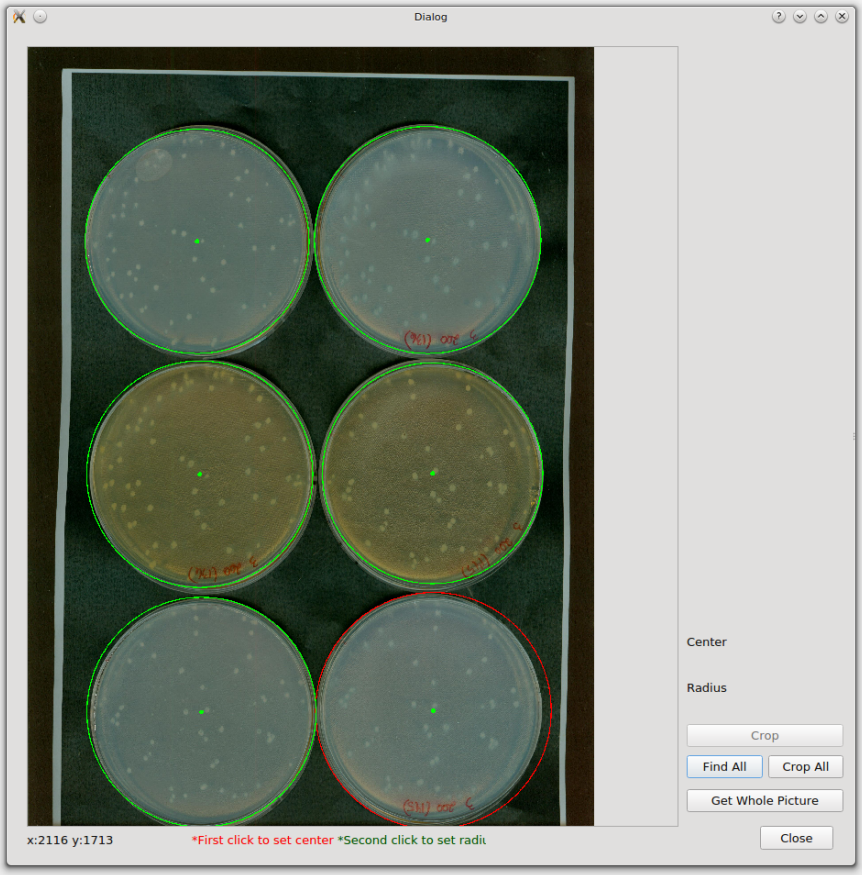


Figure 3-1: Hough Circle Transform

Since Petri dishes are always circular in shape Hough Circle Transform method was used to extract the dishes from the image in Figure 3-1, green circle means that the application can extract the whole dish while the red circle means some part of the dish are out of the image.

3.2 Segmentation

INPUT: Two cropped Petri dish

OUTPUT: Two binary images of the dishes

One of the requirements for this application is the automation of the segmentation parameters. Two approaches were used, one using an adaptive threshold using difference of Gaussian, the other was Otsu's threshold method [48]. Otsu's method work by

creating a histogram of the grey scaled version of the image which is the distribution of the pixels found in the image. A grey scaled pixel will carry a value from 0 to 255 where the zero represent the black pixel value, the 255 represent the white pixel value and any values between them will represent different shades of grey. By counting each pixel value a graph can be plotted where the x-axis will represent the pixel value and the y-axis will represent the pixel count. After plotting the image histogram the distribution of the pixel count is calculated and then a pixel value between the two dominating distribution of pixel counts will be chosen as a threshold value that will separate the objects from the background. Otsu's method turned out to work very well for some of the images, as long as the Petri dish was already located and the histogram of the images were bimodal. Images that do not have the two dominating distribution of pixel counts will result in producing a false or bad threshold value which will lead into the elimination of wanted objects or vice versa.

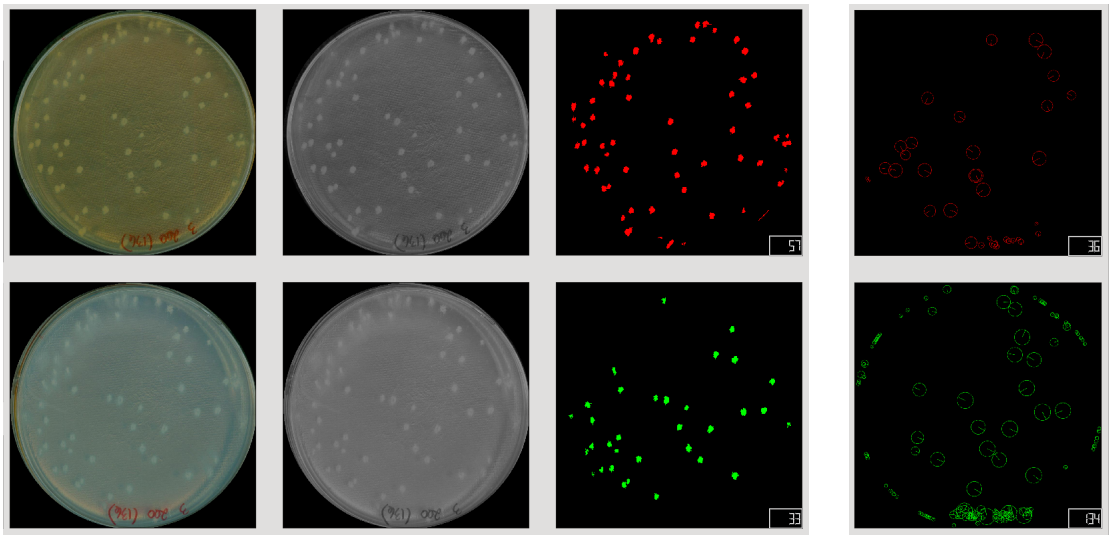


Figure 3-2: A pair of cell colonies in Manual + SURF modes

Other methods were added to guarantee best results such as manual, SURF and SIFT Figure 3-2. The manual method is a basic HSV range filter HSV stands for hue, saturation and value. The range of HSV can be expressed in 6 parameters maximum and minimum of hue, saturation, and value. The SIFT method can be used by providing 5 parameters the number of features needed, the number of octave layers, the filter contrast threshold, the edge filter threshold, and Gaussian sigma value.

The SURF method can be used by providing 3 parameters the hessian threshold, the octaves pyramid number and the number of octave layers.

Using any of these methods on an image should generate a binary image that can be used for further processing.

3.3 Cell Detection

INPUT: Two binary images of the dishes
OUTPUT: Two lists of cells positions

After applying the proper segmentation method on the two images the result will be two binary images. Each one of these binary images will contain a set of grouped pixels which in this scenario represent a cell in a binary image. To detect these cells we used Suzuki method [49] for retrieving contours from the provided binary images. After getting the contours we then calculate all the moments to the 3rd order to get the center of mass for each contours. Using these variables we can filter contours using their pixel count or by shape/curve using Douglas-Peucker algorithm [50]. From the remaining contours after filtering the unwanted shapes and sizes we can extract only the center of each cell/object and add it to a list, so each one of the two binary images will have their own list of points.

3.4 Cell Colonies Matching

INPUT: Two lists of cells positions
OUTPUT: A list of possible matching cells in pairs

3.4.1 Preliminary Experiments

Finding the corresponding points between pairs of images was the main motivation behind this research so at the beginning of this research lots of experiments were done to create a simple algorithm to find the corresponding points between two cell colonies images.

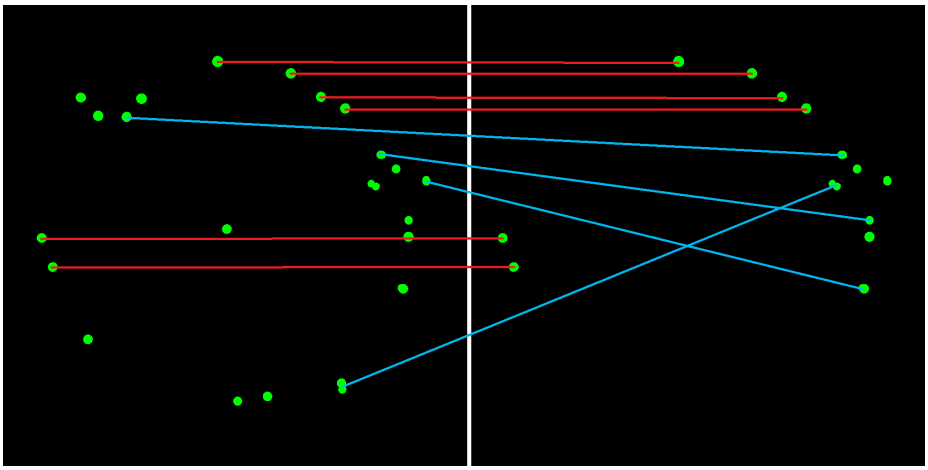


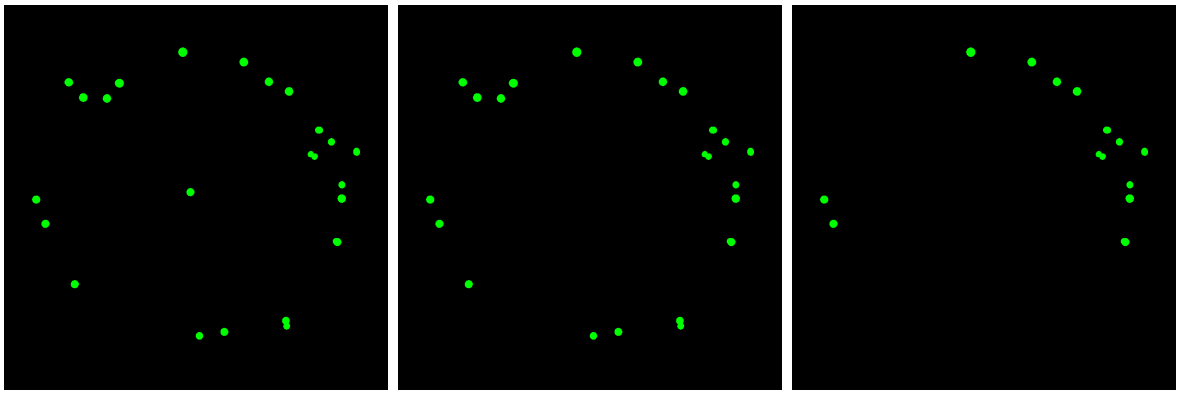
Figure 3-3: Corresponding Points Example

In Figure 3-3 the image on the right is a copy of the image of the left but with less than half of the points missing, the red lines represent correctly selected corresponding points where the blue lines represent falsely selected corresponding points.

3.4.1.1 Delaunay Triangulation

The first approach to find the corresponding point was derived from a know method called Delaunay triangulation [51] which was used to create a complex shape using all extracted points that fits a set of rules then try to match the complex shape from both images. The issue with this approach is that the complex shape will have a major change in it's shape if few points or one center point were missing.

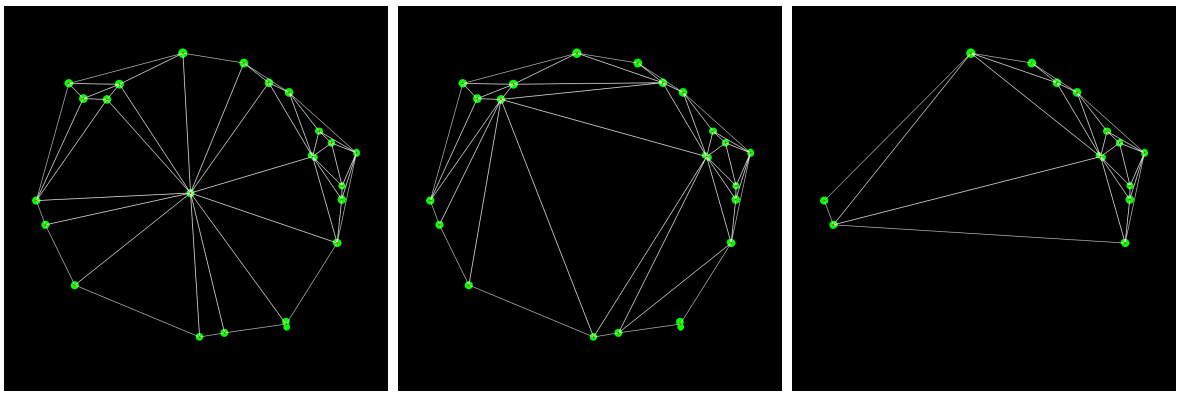
Assuming that we have the original image (a) and two possible scenarios, one with only center points missing and the other with less than half of the points missing see Figure 3-4.



(a) Original (b) Missing one center point (c) Missing less than half of the points

Figure 3-4: Example Images for Preliminary Experiment

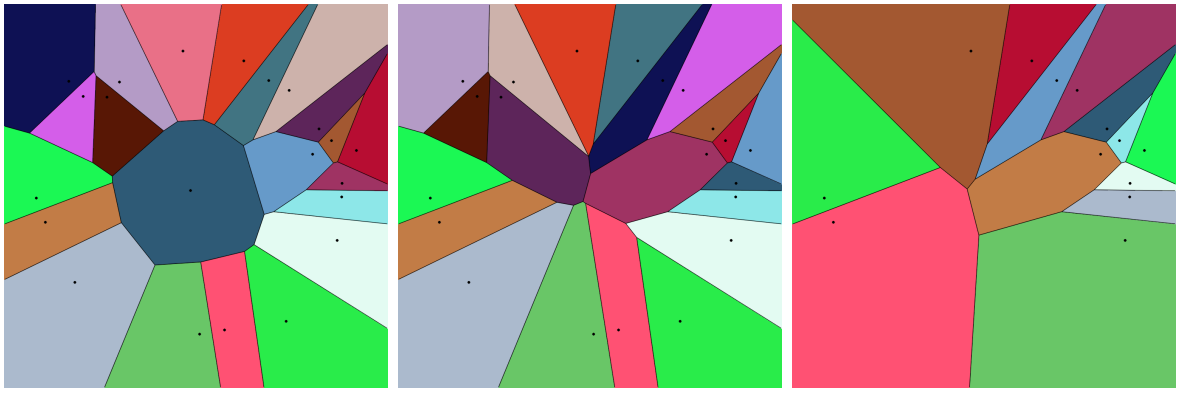
After looping through each image and extracting all points we then apply Delaunay triangulation method to all three sets of points separately. This example was chosen to demonstrate a moderate level of complexity where the image will have a major change in the left side of the shape but a pattern on the right side can still be found, but in higher complexity examples no patterns will be found using Delaunay method see Figure 3-5.



(a) Original (b) Missing one center point (c) Missing less than half of the points

Figure 3-5: Applying Delaunay Method

It is obvious that the drawn triangulations have changed dramatically but to make them easier to visualise we can use Voronoi Diagram [52] see Figure 3-6.



(a) Original (b) Missing one center point (c) Missing less than half of the points

Figure 3-6: Applying Voronoi Diagram Method

3.4.1.2 Circle Intersection Approach

By knowing the flaws of Delaunay triangulation an attempt was made to remove the weight of far or alone points that might corrupt the representation if were missing. The Delaunay triangulations can be broken down to smaller triangles by generating a new set of points from the main points and excluding the main points. Generating a new point starts with creating a circle around each point in the main points set, the circle radius will be determined by using the distance from the current point to the nearest neighbour The following pseudocode will demonstrate how we extract the radius of each point:

Listing 3.1: Circle Distance Pseudocode

```
1  INPUT: A list of cells positions
2  OUTPUT: A list of cells positions with their radius
3  Set m_points to be cells positions vector
4  Set p_size to m_points size
5  FOR i = 0 to p_size
6      Set min_dis to 1000
7      FOR j = 0 to p_size
8          IF i=j
9              Do nothing
10         ELSE
11             Set distance to Distance(m_points[i], m_points[j])
12             IF distance < min_dis
```

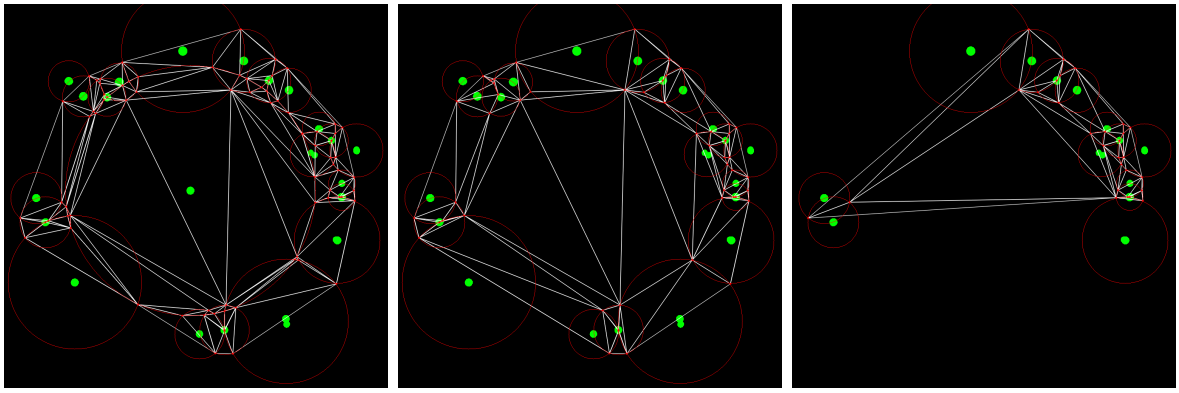
```
13                  Set min_dis to distance
14              END IF
15          END IF
16      END FOR
17      Set m_points radius to min_dis
18  END FOR
```

After extracting the radius from each point in the set shown in Listing 3.1 we will be able to generate a new set of points using the circle intersection points between every two points in the set. The following pseudocode Listing 3.2 code will clear how circle intersection points are extracted:

Listing 3.2: Circle Intersection Pseudocode

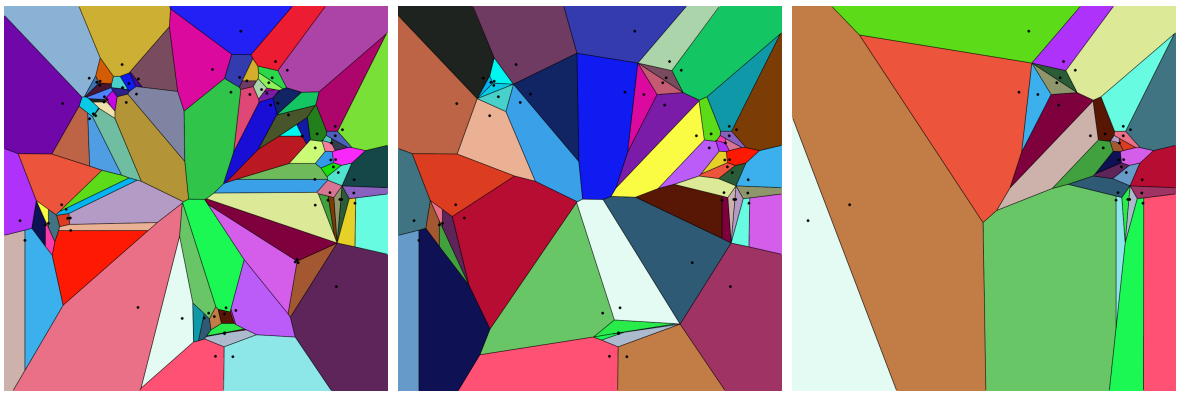
```
1  INPUT: A list of cells positions with their radius
2  OUTPUT: A list of intersections points
3  Set m_points to be cells positions & radius vector
4  Set p_size to m_points size
5  Set inter_points to be empty vector
6  FOR i = 0 to p_size
7      FOR j = 0 to p_size
8          IF i=j
9              Do nothing
10         ELSE
11             Set distance to Distance(m_points[i], m_points[j])
12             IF  distance < the sum of both radius
13                 Add a new intersection to inter_points vector
14             END IF
15         END IF
16     END FOR
17 END FOR
```

By following Listing 3.2 we are able to break the points into more points which will decrease the weight of far points that does not have a neighbour like the center point in the original image (a) in Figure 3-4. After applying the same triangulation method on the new set of points the graph became more detailed than before.



(a) Original (b) Missing one center point (c) Missing less than half of the points

Figure 3-7: Applying Delaunay with Circle Intersection Method



(a) Original (b) Missing one center point (c) Missing less than half of the points

Figure 3-8: Applying Voronoi Diagram with Circle Intersection Method

Looking at Figure 3-7 and Figure 3-8 it is clear that more patterns were preserved than the first approach but that does not mean it will work with high level of complexity. Delaunay triangulation method does not allow overlap which means it will not cover all the other possible triangulations, so it will perform badly when comparing moderate to high complexity sets of points.

3.4.2 K-NT Algorithm

The most famous techniques to find corresponding points utilize some type of triangulation, two of the known matching algorithms are Groth [2], Valdes [3]. Triangulation in geometry is the process of knowing the position of a point using the angle from the current point to another known point.

The proposed algorithm K-NT is short for k-Nearest Triangle which is based on triangulation and it is an extension of Valdes's algorithm (Section 2.5.0.2). In Valdes's algorithm only the three sides of the triangle are taken into consideration where our approach take also the neighbouring cells. The points $P1$, $P2$ and $P3$ In the following Figure 3-9 represent a triangle formed in the triangle generating stage where triangle sides are sorted based on the side length between each two points, the longest side should lay between points $P1$ and $P3$, the shortest side should lay between points $P1$ and $P2$, and the intermediate side should lay between points $P2$ and $P3$ as shown in Figure 3-9.

Instead of processing only vertexes $P1$, $P2$ and $P3$, we also take the closest n neighbours where n is a user specific parameter Figure 3-9. $P1$ should lay between the longest and intermediate sides so the closest neighbours will be the same in other images. We calculate the Euclidean distance between point $P1$ and neighbours Nn and sort in ascending order. After sorting the closest neighbours we then select the first n neighbours and store them in a list.

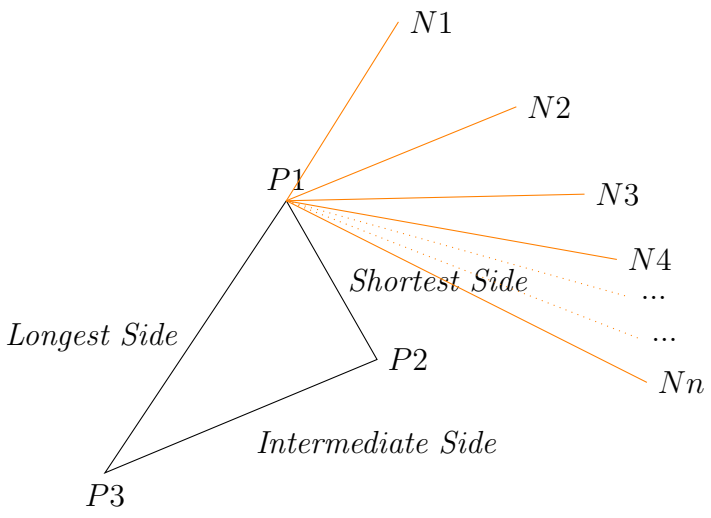


Figure 3-9: Nearest Neighbour N

After creating the list of the nearest neighbours we then transform the triangle points into the triangle space as what Valdes's does in his algorithm. However, we also store the list of the nearest neighbours alongside the triangle space coordinate shown in Listing 3.3.

Listing 3.3: K-NT Triangles Generating Pseudocode

```
INPUT: image one and image two blobs positions
OUTPUT: two collections of generated triangles with their neighbours
Set p_image_1   to be a vector of all cells positions in image one
Set p_image_2   to be a vector of all cells positions in image two
Set triangle_c1 to a vector where triangles will be stored for image one
Set triangle_c2 to a vector where triangles will be stored for image two
Set n           to be a user defined considered neighbours count
FOR each triangle combination in each set (p_image_1, p_image_2)
    Set v1 to be the vertex between shortest and longest side of triangle
    Set v2 to be the vertex between shortest and intermediate side of triangle
    Set v3 to be the vertex between longest and intermediate side of triangle
    Set x  to be triangle intermediate side length / longest side length
    Set y  to be triangle shortest side length / longest side length
    Set list_n to be a vector to store neighbours distances
    Set list_n_top to be a vector to store top neighbours distances
    For each point in either p_image_1 or p_image_2
        Store euclidean_distance(v1, point) into list_n
    END FOR
    Sort list_n by distance
    For each distance in list_n up to n
        Store distance/a into list_n_top
    END FOR
    Store x, y, v1, v2, v3, list_n_top in triangle_c1 or triangle_c2
END FOR
```

We match the two lists using the Euclidean distance and the rate of error. The rate of error is what differ our algorithm from Groth's and Valdes's, we produce the rate of error by calculating the total absolute difference between n neighbours shown in equation 3.1

$$Error = \sum_{i=1}^{n} |a_i - b_i| \tag{3.1}$$

where a and b are two points in the triangle space each from a separate lists and n represent the number of neighbour taken in consideration.

After calculating the rate of error we then add it to the distance between the two points from separate lists. A perfect neighbours match should make the rate of error close to zero which will not affect the distance calculated between the two points. If we have a small Euclidean distance between two points but a large rate of error, the final distance will be the sum of both which will give us a fair assessment of the match Listing 3.4.

Listing 3.4: K-NT Triangles Matching Pseudocode

```
1  INPUT: two collections of generated triangles with their neighbours
2  OUTPUT: a collection of triangles with their possible matches
3  Set triangle_c1 to be a vector of triangles with their neighbours (Image one)
4  Set triangle_c2 to be a vector of triangles with their neighbours (Image Two)
5  Set n           to be a user defined considered neighbours count
6  FOR each tri_1 in triangle_c1
7      FOR each tri_2 in triangle_c2
8          Set n_error_rate to be 0
9          FOR i = 0 to n
10             ADD abs(tri_1.list_n_top[i], tri_2.list_n_top[i]) to n_error_rate
11         END FOR
12         Set distance to be euclidean_distance(tri_1, tri_2) + n_error_rate/10
13         IF distance < tri_1.current_match_distance
14             Store distance into tri_1.current_match_distance
15             Store tri_2 address into tri_1.current_match
16         END IF
17     END FOR
18 END FOR
```

After processing the matches we then sort them by the distance (which should include the rate of error and the Euclidean distance) in ascending order. After sorting them by distance we select only the top S which is a user specific parameter. In our testing we have chosen the number of neighbours (n) to be 5 and 8 and we use the top 10 matches (S) to evaluate the algorithms.

3.4.3 An Example of Cell Colonies Matching

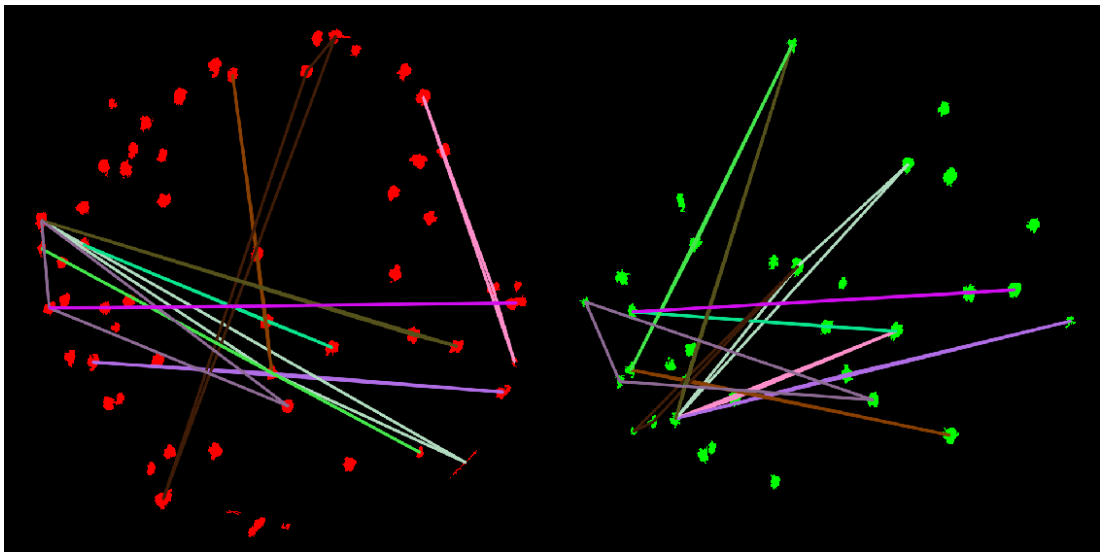


Figure 3-10: The top 10 matching triangles using Valdes's algorithm.

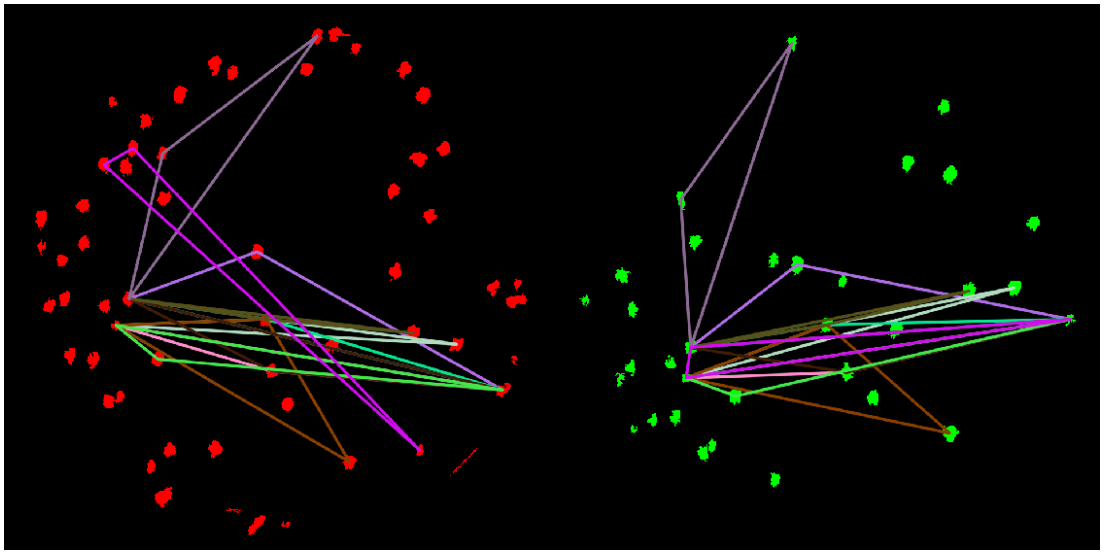


Figure 3-11: The top 10 matching triangles using K-NT n=5.

In Figure 3-10 and Figure 3-11 the red colour represent the blobs of the first dish while the green colour represent the blobs second dish, also triangles with the same colour in both dish one and two represent a possible match. We can observe that Valdes algorithm is correctly matching few of the corresponding points Figure 3-10 where the K-NT is correctly finding most of the corresponding points Figure 3-11.

3.5 Image Alignment

INPUT: A list of possible matching cells in pairs

OUTPUT: An aligned image

3.5.1 OpenCV

OpenCV offers two algorithms to estimate a model from a set of data that might have an outliers. The first one is called Random Sample Consensus known as RANSAC is an algorithm published by Fischler and Bolles [53]. The second onw is called Least Median of Squares known as LMedS is an algorithm published by Peter J [54].

These algorithms is used by OpenCV to estimate the perspective transformation which requires at least three pairs of points from each list, each pair of points should represent a possible match between them so that pair number one from list one matches pair number one from list two. To make the estimation better one must provide at least 8 correctly matched points which in our example can be expressed as two correctly matched triangles. Since OpenCV implementation only accept pairs of points not pairs of triangles we need to extract the points from the triangles before passing them to the OpenCV model estimation algorithm. From each triangle 3-12 points (A), (B), and (C) can be extracted directly from the stored triangle data structure but any other points can be mathematically extracted as long as the extracted point position is extracted using the ratio of the main three points stored. In our example we have extracted the center of the triangle (D) by using the centroid of triangle 3.2.

$$D_x = \frac{A_x + B_x + C_x}{3}, \quad D_y = \frac{A_y + B_y + C_y}{3} \tag{3.2}$$

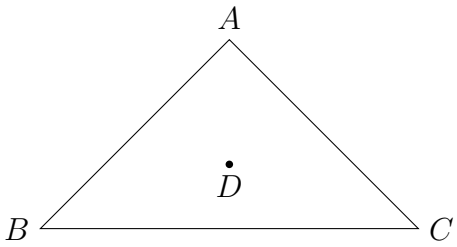
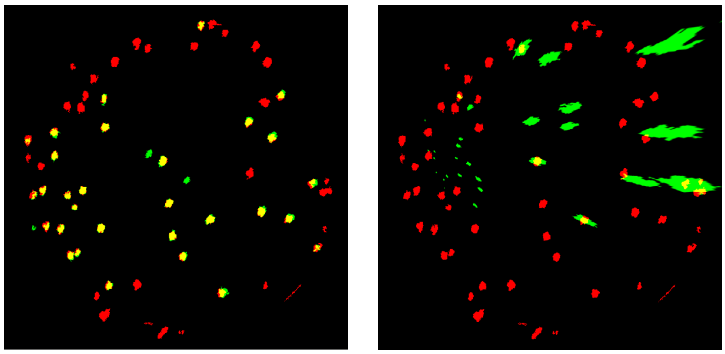


Figure 3-12: Triangle Points

Using the top 10 matches (S) from the proposed algorithm we can at least extract 30 matching pairs of points or 40 if we include the center of each triangle. The result of these algorithms will be a transformation matrix.

3.5.2 Automatic Alignment Evaluation

After the application compute the top (S) matching triangles from each list using K-NT algorithm, one of the robust estimators algorithms will be used either RANSAC or LMEDS to create the transformation matrix. The transformation matrix will be applied to do a geometric rotation, translation or affine transformation to one of the two images (A, B) taken from the Cell Detection stage. By applying the matrix to one of the images (A) for example it will produce a transformed image that can be compared to the other image (B) by adding the new image on top of the other image (B).



(a) (b)

Figure 3-13: The alignment after applying RANSAC to (a) K-NT n=5 triangle and (b) Groth's matching triangles.

In Figure 3-13 the red and green pixels represent the original blob parts for each image, and the yellow pixels their intersection after the alignment. We created a convention that one image blobs are red and the other are green and their intersection will be yellow. The yellow colour is the mix of the two colours in this example it is the combination of red and green thus it will produce yellow as the third colour. The yellow colour will be visible when two cells from each image is aligned correctly which

give us the some rules which can indicate the status of the alignment the rules are as followed:

1. If the new combined image has only yellow coloured cells that indicate the images were perfectly aligned.

2. If the new combined image has only red or green coloured cells that indicate the images were unsuccessfully aligned.

3. If the new combined image has only red and green *without the yellow* that indicate the images were unsuccessfully aligned.

4. If the new combined image has all three colours at once then it can be either aligned or otherwise.

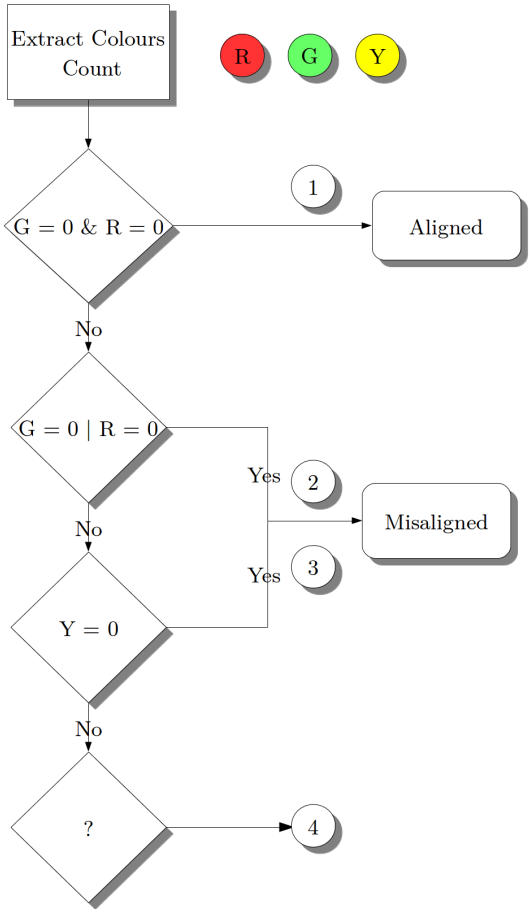


Figure 3-14: Flow chart of the current alignment rules

These rules can be programmed to notify the user if the alignment were successful or not except that the last rule can not be determined since there are many scenarios that can not be covered by our observation. In order to have an alignment indicator in the application the last rule must be further investigated. In the alignment stage it is possible to extract 10 useful attributes that might help in determining the status of the alignment. The machine learning features are as follow:

1. The red cells count before the alignment (Image one).
2. The red pixels count before the alignment (Image one).
3. The red cells count after the alignment (Image one).
4. The red pixels count after the alignment (Image one).
5. The green cells count before the alignment (Image two).
6. The green pixels count before the alignment (Image two).
7. The green cells count after the alignment (Image two).
8. The green pixels count after the alignment (Image two).
9. The yellow cells count (Combined image).
10. The yellow pixels count (Combined image).

These extracted attributes will be used with several machine learning algorithm to train a classifier that will classify if two images are aligned correctly or not, using the the mentioned 10 attributes.

Chapter 4

Results and Discussion

Hundreds of images containing 1-3 pairs of Petri dishes were acquired according to the needs of the Biology research group. Colonies with 50, 100 and 200 cells were tested for this work. In this section we demonstrate the accuracy of the proposed algorithm K-NT compared to other matching algorithms and show the AdaBoost classifier accuracy as well.

4.1 Matching Algorithms

To test the performance of K-NT matching algorithm compared to other matching algorithms, namely BFM, FANN, Groth, Valdes which we evaluate in two stages. After performing the matching process between 60 pairs of images containing the cell colonies, the top 30 matches for each pair of images were extracted to clearly identify the best combination of algorithms for matching cell colonies. The first stage is how many corresponding points out of the top 30 was correctly matched for each one of the matching algorithm. The second stage is about the alignment, so for each matching algorithm, we count the number of correctly aligned colonies. If enough corresponding points were found correctly, it should be straightforward to align the images. However, because the location of the points depend on the quality of the segmentation steps, the alignment may fail. For SURF and SIFT, due to the nature of their operations that rely on a multitude of features, such as size, angle, etc., cell colony matching

was observed to be unsuccessful. As compared to triangulation methods, it shows that looking at cell position is the key feature to successful matching. In combination with SIFT and SURF, we tested two options, Fast Approximate Nearest Neighbour Search (FLANN [55]) and a brute force method (called BFM in OpenCV [47]). For completion of the comparison process, we used the location of features extracted using SIFT and SURF descriptor as points for the triangulation methods.

Before automating the process completely, we wanted to see how sensitive the final alignment was to specific parameters for each method. After many attempts, we collected the best results for each batch. For SURF, we experimented with the following parameters: Hessian threshold (Hess), number of pyramid octaves (NPC) and number of octave layers (NOL). For SIFT: Number of best features to retain (NBFR), number of layers in each octave (NLO), contrast threshold (CT), edge threshold (ET) and Gaussian sigma (GS) (please refer to Table 4.1).

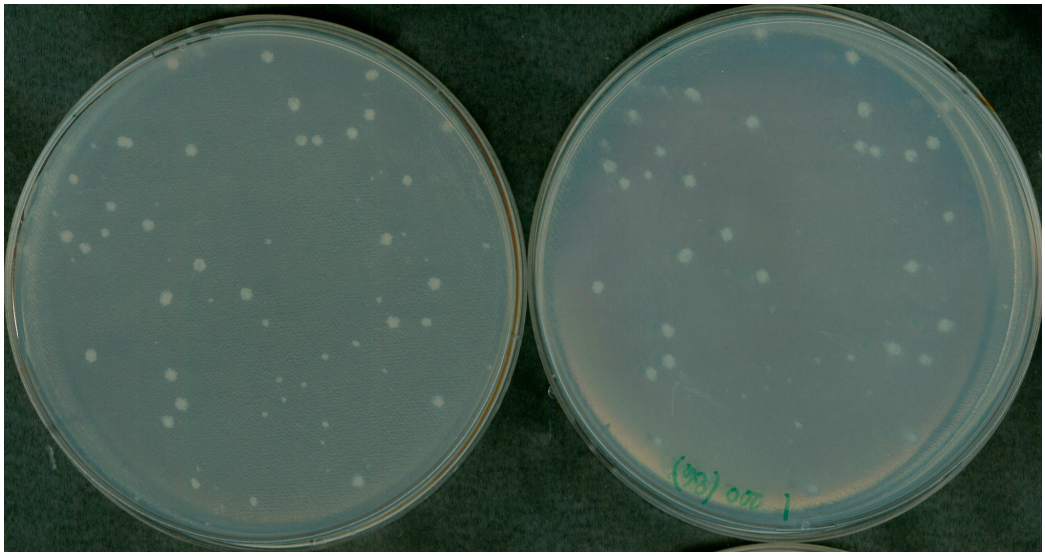


Figure 4-1: One of the pairs of images used in the experiments. Refer to Table 4.1 for SURF and SIFT settings.

Multiple difficult pairs of images were tested e.g. (Figure 4-1). To try to make SURF and SIFT work, we need to change their adjustable parameters. After adjusting these parameters, the alignment was carried out, either via one of the triangulation methods, or via FLANN or BFM. The parameters used are listed in Table 4.1.

		SURF			SIFT				
		Hess	**NPC**	**NOL**	**NBFR**	**NLO**	**CT**	**ET**	**GS**
pair	image 1	1000	1	6	181	1	0.04	0.45	5
	image 2	720	1	8	181	4	0.04	0.45	5

Table 4.1: Best parameters for SIFT and SURF.

Methods		**Top 30 Points Matching**		**Alignment Algorithms**	
Features	**Matching**	**Incorrect Matches**	**Correct Matches**	**RANSAC**	**LMEDS**
Binary	Groth	15	15	TRUE	TRUE
	Valdes	6	24	TRUE	TRUE
	K-NT n=8	0	30	TRUE	TRUE
	K-NT n=5	0	30	TRUE	TRUE
SURF	Groth	30	0	FALSE	FALSE
	Valdes	27	3	FALSE	FALSE
	K-NT n=8	6	24	TRUE	TRUE
	K-NT n=5	0	30	TRUE	TRUE
	FLANN	30	0	FALSE	FALSE
	BFM	30	0	FALSE	FALSE
SIFT	Groth	30	0	FALSE	FALSE
	Valdes	30	0	FALSE	FALSE
	K-NT n=8	30	0	FALSE	FALSE
	K-NT n=5	30	0	FALSE	FALSE
	FLANN	30	0	FALSE	FALSE
	BFM	30	0	FALSE	FALSE

Table 4.2: SIFT & SURF performance example

The Table 4.2 shows an example of how SURF or SIFT are performing and due to such bad performance it was excluded from the final testing summarised in Table 4.3. This was somewhat expected for this type of pair of images, as the cell colony blobs do not have a specific shape and may be very similar to each other.

	K-NT n=5	**Valdes**	**Groth**
TRUE Triangle Matches	79.5%	24.3%	20.2%
FALSE Triangle Matches	20.5%	75.7%	79.8%
Correctly aligned by LMEDS	81.7%	16.7%	11.7%
Correctly aligned by RANSAC	98.3%	46.7%	33.3%

Table 4.3: 60 pairs results

The Table 4.3 summarise the results obtained with 60 pairs of images from the data set which can be replicated. Due to the possible choice of parameters, we manually adjusted the HSV range until we got the best possible segmentation that will work for all algorithms and then saved the parameters.

The results clearly show (Table 4.3) that K-NT n=5 method are almost 4 times better when matching triangles as compared to the original Groth [2] and Valdes [3]. The former algorithms work very well for Astronomy, where extra parameters such as the brightness of stars can be used to set the initial points for the triangulation process. However, in our images corresponding blobs can have very different sizes and shapes, and several blobs may be missing in one of the images.

When using the K-NT method with binary blobs, a good alignment was always found with both RANSAC and LMEDS. There were cases where either Groth or Valdes failed to find a good alignment.

4.2 The Proposed AdaBoost Classifier

Since the alignment stage (Section 3.5.2) can produce 10 useful attributes per alignment attempt an effort was made to collect the attributes from more than 250 alignments attempts. Each one of these alignment attempt was manually classified as correctly aligned or misaligned and then was given to several classification supervised and unsupervised algorithms such as AdaBoost, Neural Network, C4.5, etc. By using the 10-fold cross-validation method to evaluate the algorithm it showed that none of the algorithm was able to achieve more than 80% accuracy. The algorithms were

performing badly because the attributes were not normalised and to fix the issue some attributes were used in the normalisation process, for example, the green pixel count was divided by the yellow pixel count and the red pixel count was also divided by the yellow pixel count to produce a normalised values see equations (4.1). After normalising the attributes a second attempt at classification algorithms was done and evaluated which impacted the accuracy greatly by adding 10%-20% accuracy to the previous results.

$$
\begin{aligned}
AA &= \text{“After Alignment”} \\
\text{Red \& Yellow Counts Ratio AA} &= \frac{\text{Yellow Cells Count AA}}{\text{Red Cells Count AA}} \\
\text{Green \& Yellow Ratio} &= \frac{\text{Yellow Pixel Count}}{\text{Green Pixel Count}} \\
\text{Green \& Yellow Counts Ratio AA} &= \frac{\text{Yellow Cells Count AA}}{\text{Green Cells Count AA}} \\
\text{Red \& Green \& Yellow Ratio} &= \frac{\text{Yellow Pixel Count}}{(\text{Green + Red Pixel Counts})/2} \\
\text{Red \& Green \& Yellow Ratio AA} &= \frac{\text{Yellow Pixel Count AA}}{(\text{Green + Red Pixel Counts AA})/2}
\end{aligned}
\tag{4.1}
$$

4.2.1 Further Optimizing the AdaBoost Results

The algorithm that scored the highest accuracy was AdaBoost which achieved the accuracy of 98.5% Table 4.4 using the 10-fold cross-validation technique that means 263 out of 267 were correctly classified Table 4.5. Since AdaBoost work by combining multiple weak learners (Section 2.6) in this example AdaBoost was able to use only 10 weak learners which can be combined to produce the accuracy of 98.5%. The 10 weak learners are simply a decision stumps so it can be read easily thus sometimes

it can be optimised even further. By examining all the decision stumps it appears that some decision stumps share the same classifier line (vertical or horizontal) but has different weights so combining these decision stump will make the classifier more optimised (speed wise). By optimising the classifier we are able to cut down the required input for the classifier to only 5 normalised attributes instead of the original 10 normalised attributes.

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**	**Class**
	0.979	0.008	0.993	0.979	0.986	0.991	Aligned
	0.992	0.021	0.977	0.992	0.984	0.991	Misaligned
Weighted Avg	0.985	0.014	0.985	0.985	0.985	0.991	

Table 4.4: AdaBoost Accuracy Details

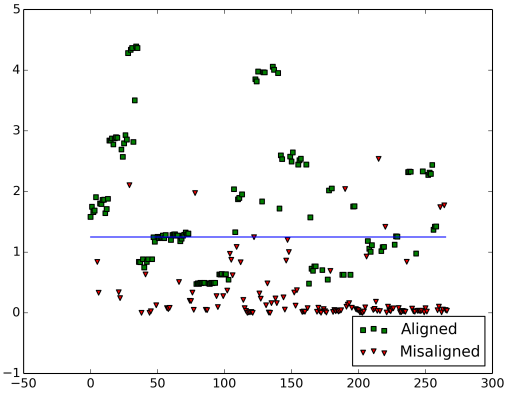
	Aligned	**Misaligned**
Aligned	137	3
Misaligned	1	126

Table 4.5: Confusion Matrix

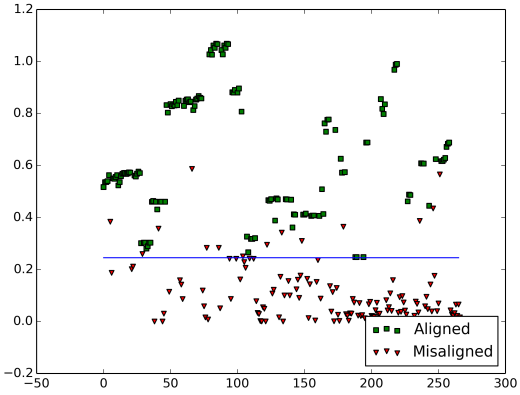
4.2.2 Weak Learners

The goal of using a classification algorithm is to train a classifier that will help to identify if an alignment is correct or not. AdaBoost weak learners are a simple decision stumps which determine where the weight will be assigned, so if an attribute value is larger than the learner threshold then the weigh if that learner will be assigned to either correctly aligned or misaligned variable. After doing the same steps for all the weak learners the variable that has the highest value will be used to label the alignment stage to either successfully or unsuccessfully aligned. For example, if we take the weak learner 1 in Figure 4-2 which uses the attribute named "Red & Yellow Count Ratio After Alignment" which means that the attribute is normalised by dividing the red cells count by the yellow cells count in the alignment image. In the Figure 4-2 if the attribute of weak learner 1 "Red & Yellow Count Ratio After Alignment" value is more than the threshold (1.2491860691) a weight of 1.24 will be

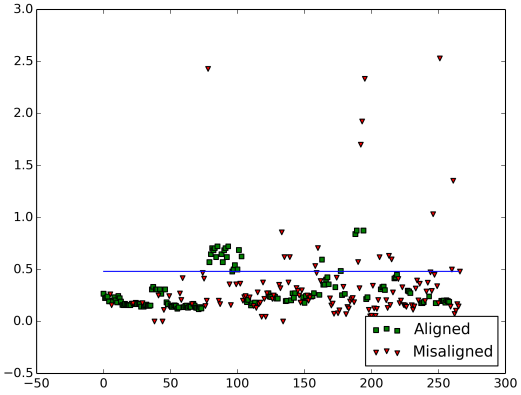
assigned to the correctly aligned variable; on the other hand, if the attribute value is less than the threshold the weight will be assigned to the misaligned variable. By doing the mentions steps to the remaining weak learners in Figures 4-2, 4-3 the AdaBoost classifier will produce the accuracy of 98.5% in classifying the aligned image.



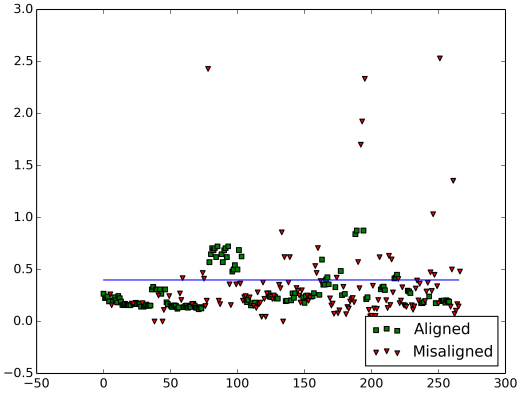
Weak Learner 1: Red & Yellow Counts Ratio After Alignment



Weak Learner 2: Green & Yellow Ratio

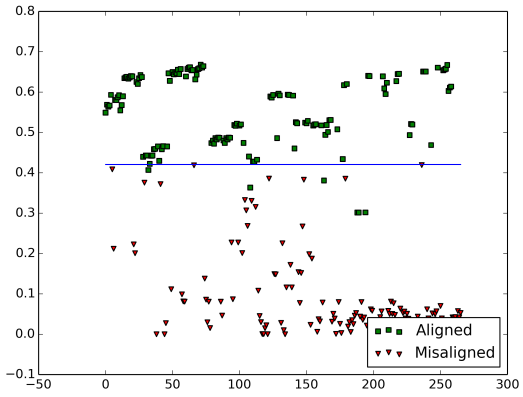


Weak Learner 3: Green & Yellow Ratio Counts After Alignment

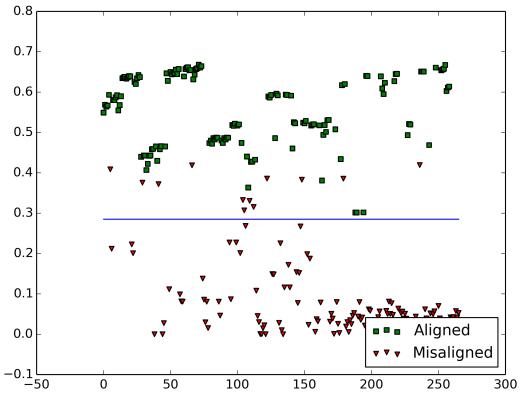


Weak Learner 4: Green & Yellow Ratio Counts After Alignment

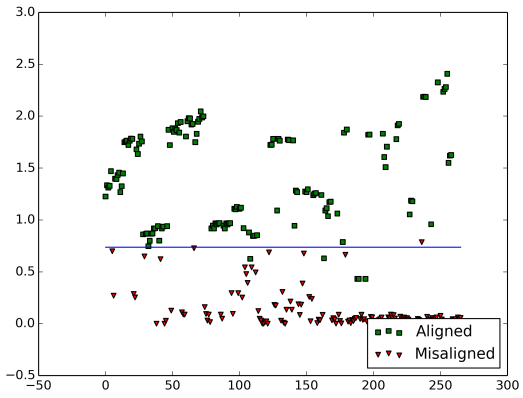
Figure 4-2: Classification using Red & Yello Ratio Values + Green & Yellow Ratio Values



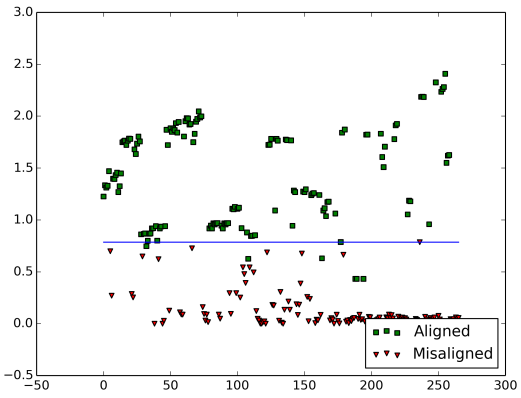
Weak Learner 5: Red & Green & Yellow Ratio



Weak Learner 6: Red & Green & Yellow Ratio



Weak Learner 7: Red & Green & Yellow Ratio After Alignment



Weak Learner 8: Red & Green & Yellow Ratio After Alignment

Figure 4-3: Classification using Red & Green & Yellow Ratio Values

Using the figures above and each threshold we can generate a simple code in C++ that will classify if the given attributes are from an aligned image or from a misaligned one.

Listing 4.1: Optimized C++ AdaBoost Classifier Code

```
void AdaBoost::Evaluate(float green_yellow_count_after, float green_yellow,
                        float red_green_yellow_after,
                        float red_green_yellow, float red_yellow_after)
{
    A=M=0;
    (red_green_yellow > 0.4206722775)          ?  A+= 4.50 : M += 4.50;
    (red_green_yellow> 0.28496798155)          ?  A+= 4.58 : M += 4.58;
    (green_yellow_count_after > 0.48074074075) ?  A+= 1.30 : M += 1.30;
    (red_green_yellow_after > 0.7367843402)    ?  A+= 1.18 : M += 1.18;
    (green_yellow > 0.24484761465)             ?  A+= 1.38 : M += 1.38;
    (red_yellow_after > 1.2491860691)          ?  A+= 1.24 : M += 1.24;
    (green_yellow_count_after > 0.39811912225) ?  A+= 1.17 : M += 1.17;
    (red_green_yellow_after > 0.7864564795)    ?  A+= 0.70 : M += 0.70;
    A /= 16.05;
    M /= 16.05;
}
```

The code in Listing 4.1 is an actual code taken from the final project source code. The function Evaluate is a part of class called "AdaBoost" that was designed to handle the classification part. The function take only the five needed attributes and it will assign the weight of each weak learner to either variable Aligned (A) or Misaligned (M). After assigning the weights a simple division by the total weight is done to get the accuracy percentage and the variable with the bigger number will decide if the given attributes indicate an alignment or misalignment.

Chapter 5

The Complete System: Software Architecture & Usage

The research goal was to create an application that will help the biology department to automate the pairs of Petri dishes matching process. The combination of Qt framework for GUI and C++ for the logical part will make the code able to run on the major operating systems. We have used multiple C++ classes to simplify the solution and also to make it easy for later adaptation of part of the application.

The following class diagram in Figure 5-1 shows the simple relations between classes.

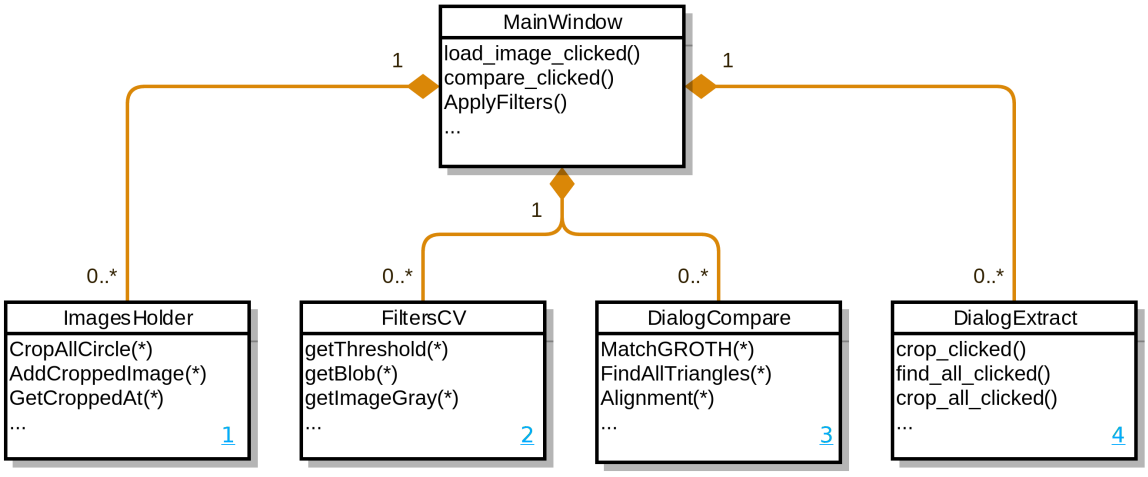


Figure 5-1: Software Architecture Overview

5.1 MainWindow

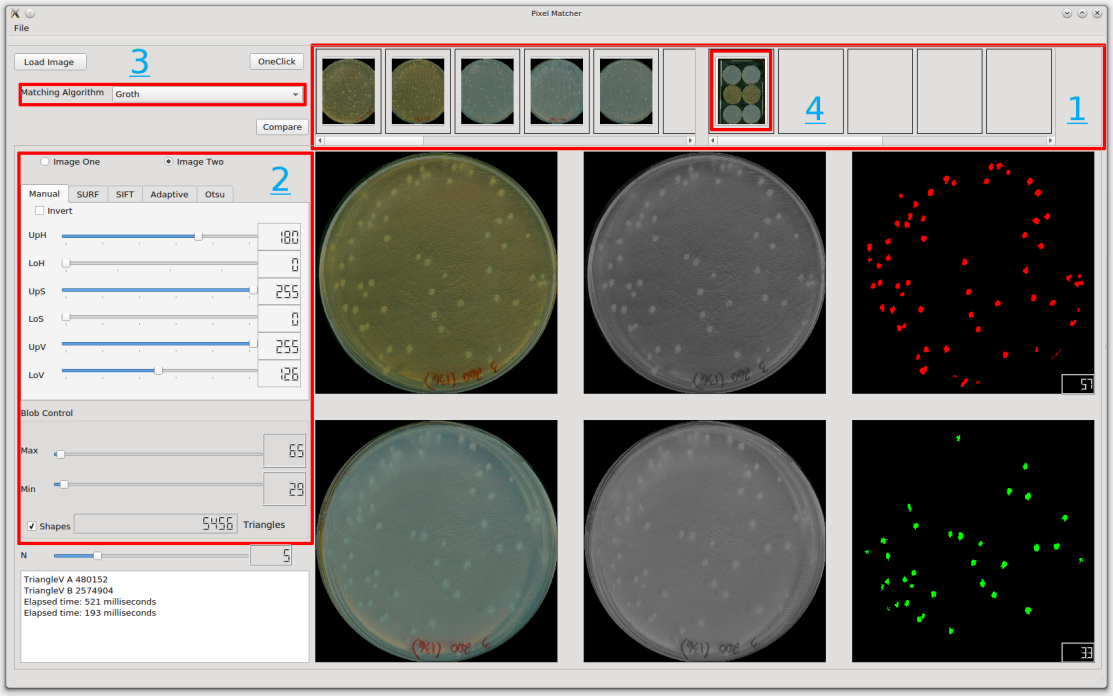


Figure 5-2: MainWindow Class GUI: The numbers represent which class (Figure 5-1) handles the specific area.

The MainWindow class represent the main GUI that contains every other class. The purpose of this class is to connect the logical part with the presentation part. In this class many things can be done such as, automatically generating the GUI elements from a list of strings, setting up a button event, or initialising other classes at run time.

5.2 ImageHolder

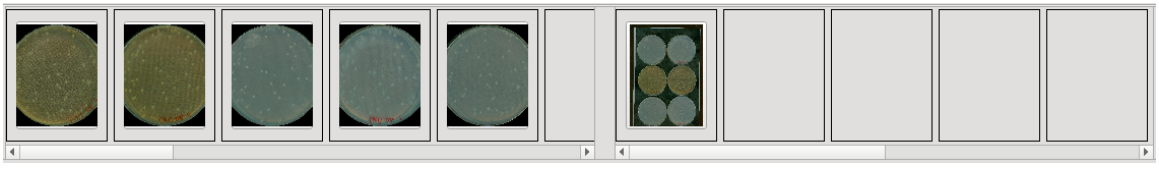


Figure 5-3: ImageHolder Class GUI

ImagesHolder class is responsible for representing the image and it's geometry as well as acting as a container of the loaded images. This class have methods that can

manipulate the image to remove unwanted areas and focus on the important part, these methods can perform what is known as image cropping and since we have an idea of what is important in an image; a Hough Circles [47] function was implemented to make the selection of Petri dishes automated as an option. The class contains two vectors one to hold the loaded images and one to hold the cropped ones which are being displayed to the user Figure 5-3 , also it has two temporary image holders to hold the two current cropped images that is being processed by other classes. Every one of the two temporary images are overwritten when the user click a new cropped image from the list of cropped images left list seen in Figure 5-3.

5.3 DialogExtract

Figure 5-4: DialogExtract Class GUI

After loading the image using the "Load Image" button, the user then can click on the image from the list in Figure 5-3 to start cropping the image or directly loading it to the cropped image container if the image is already has been cropped by an external application. By clicking the image the class DialogExtract will be called which will initialise a dialog GUI. The dialog GUI can be described to be the front end of the ImageHolder class methods where the GUI will enable the user to use the cropping methods.

5.4 FiltersCV

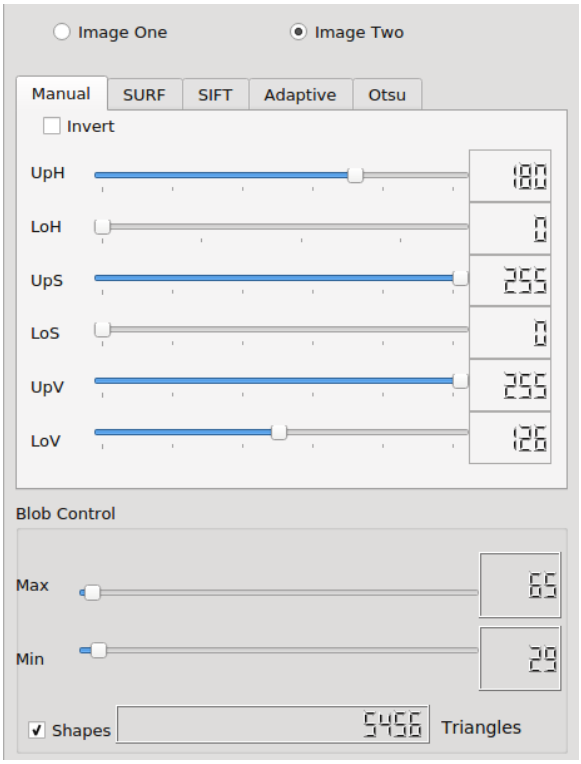


Figure 5-5: FiltersCV Class GUI

FiltersCV class is one of two essential parts of this application since it handles all colour related image processing. By changing any value of the GUI elements in Figure 5-5 an event will be triggered to call the appropriate function. FiltersCV contains functions that are needed before applying any of the matching algorithms. Some of the functions are listed below:

- Convert image colours to grey-scaled.
- Extract a binary representation of an image.
- Extract the image features using either SIFT or SURF (Section 2.3).
- Apply either adaptive or Otsu thresholds (Section 3.2).
- Detecting blobs in an image with the ability to filter unwanted blobs sizes or shapes.

- Check if the information gathered meets the minimum requirements to apply some matching algorithms i.e., FLANN & BFM matching algorithms (Section 2.4) works only with SIFT & SURF features extractors.

5.5 DialogCompare

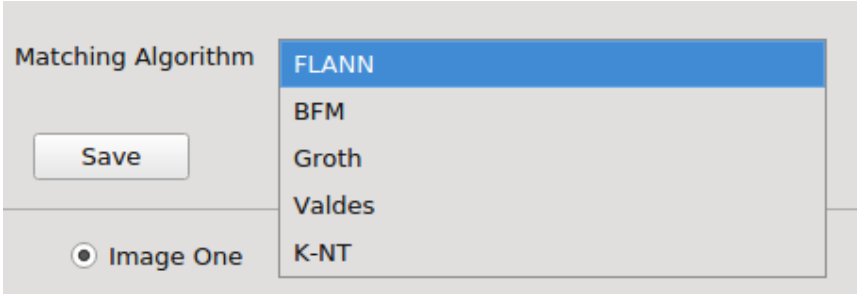


Figure 5-6: DialogCompare Class GUI

The DialogCompare class is the other essential part of this application because it handles the corresponding points matching as well as the alignment part. This class has the implementation of Groth [2], Valdes [3], FLANN, BFM, and the proposed algorithm which are broken into two functions for each algorithm. The first function is designed to find all possible triangles using one of the selected matching algorithm method. The second function is designed to match the generated triangles from the first function using the selected matching algorithm specific method.

After matching the two images the user is then able to perform the alignment using either RANSAC[53] or LMEDS[54] estimator algorithms. The application will automatically run the AdaBoost classifier to classify if the resulted image of the alignment algorithm is aligned or misaligned.

5.6 System Walk-through

When the user runs the application the main GUI will appear Figure 5-7 but it will have most buttons disabled (unclickable). To proceed the user needs to click the button "Load Image" which will promote a file explorer dialog Figure 5-8 which will

let the user choose any image with the format PNG, xpm, JPG or BMP (additional formats can be easily enabled in the code).

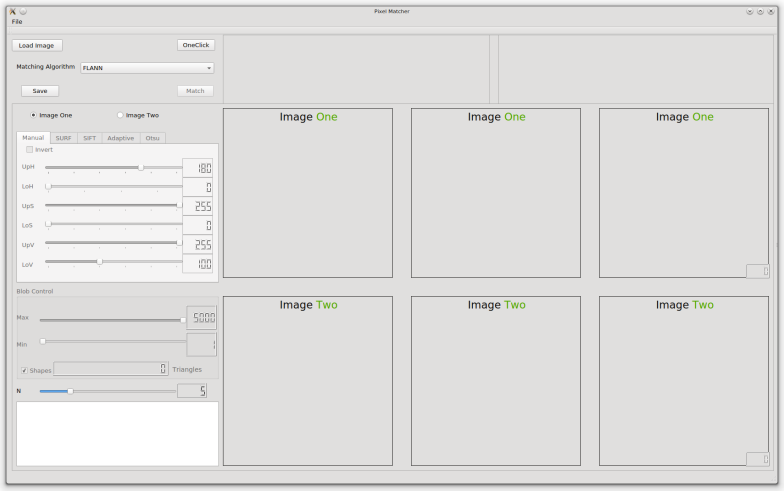


Figure 5-7: The Main GUI

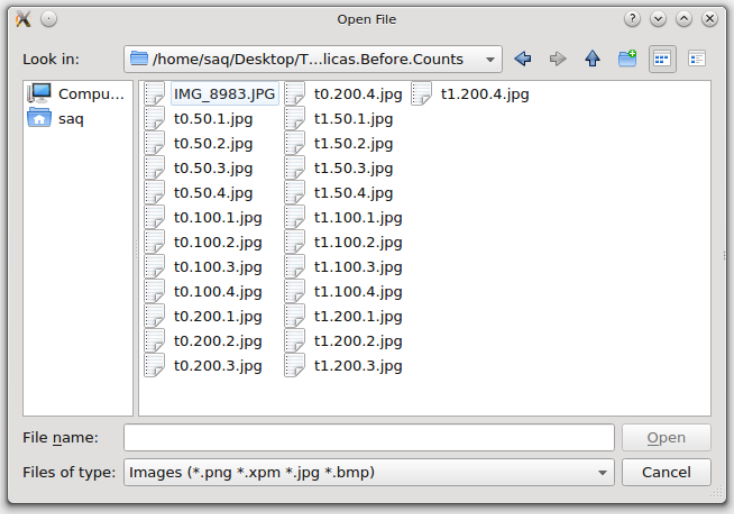


Figure 5-8: Load Image Dialog GUI

After the user select the image that contains the Petri dishes they will need to click on the image thumbnail that was added when the File Dialog was closed. When the user click on the thumbnail of the image it will promote the extraction dialog which will enable the user to either manually select and crop a Petri dish (left dialog in Figure 5-9) or use the automated the Hough Circle [47] feature to find all dishes and then crop them (right dialog in Figure 5-9).

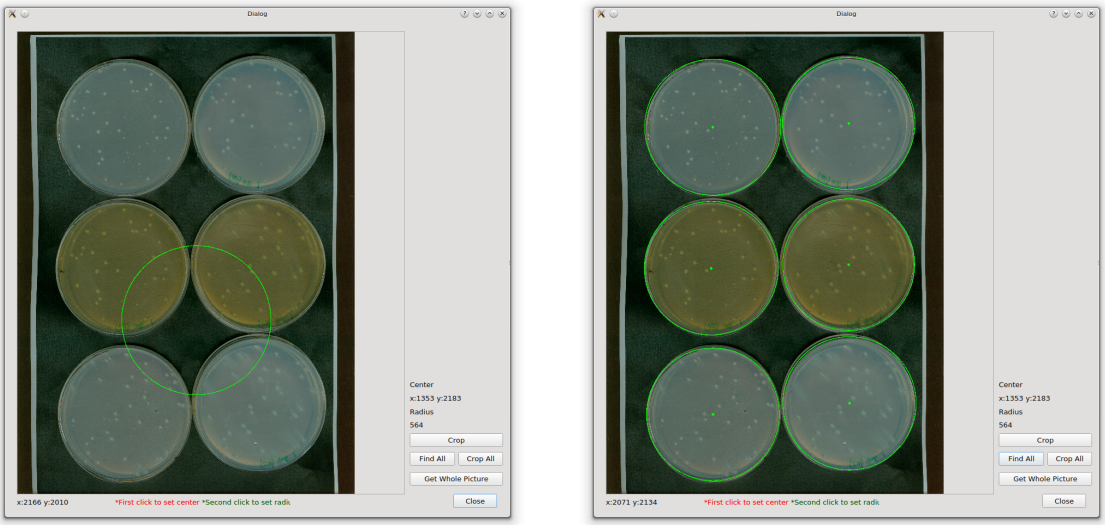


Figure 5-9: Image Cropping Dialog GUI

Every cropped dish will be loaded into the GUI cropped images list and will be assigned a thumbnail, when the user click on the cropped image thumbnail it will be either loaded into the main image holder one or two depending on what the radio button is set to. After loading the two cropped images the filtering toolbox will be enabled for the user to use Figure 5-10. The user can skip the filtering process and use the the blobs detection slider Figure 5-11.

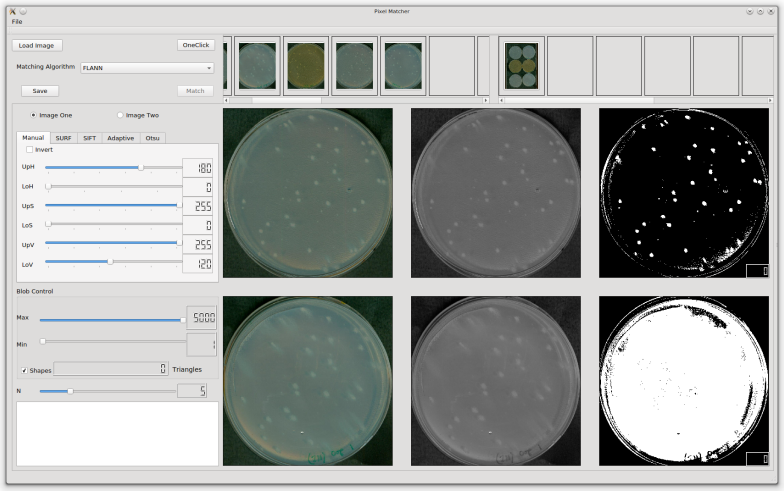


Figure 5-10: Filtering Toolbox GUI

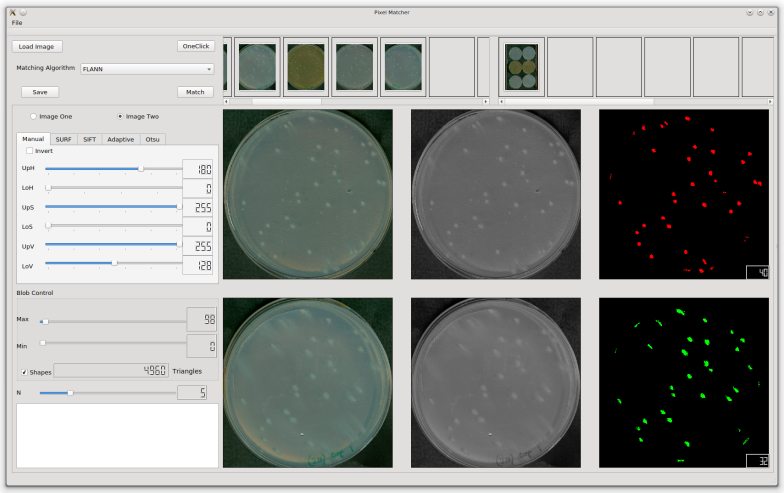


Figure 5-11: Blobs Detection GUI

The user can set the maximum and the minimum blob size by using the sliders under "Blob Control" toolbox Figure 5-11 and also the user have the ability to turn on the shape filtering when detecting blobs which should help when the cells size are too small. After extracting the blobs from both images the user can choose the wanted matching algorithm and click the button "Match".

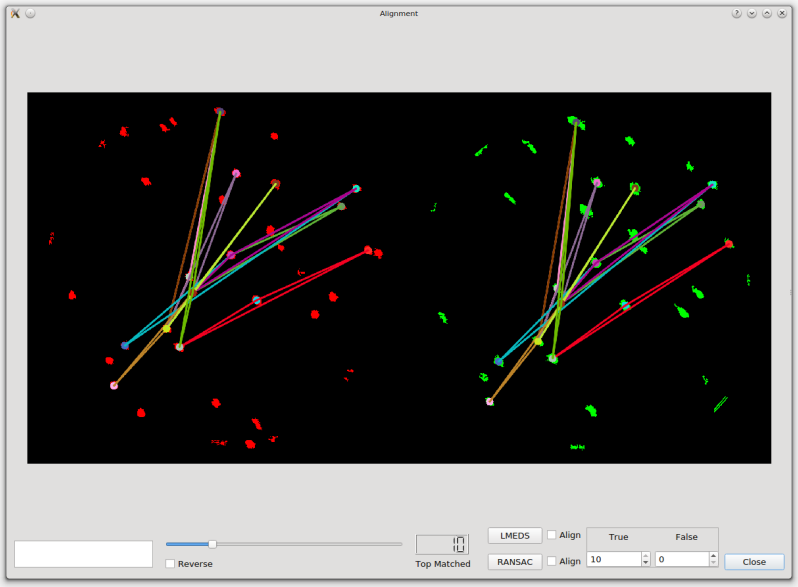


Figure 5-12: K-NT Matching GUI Top

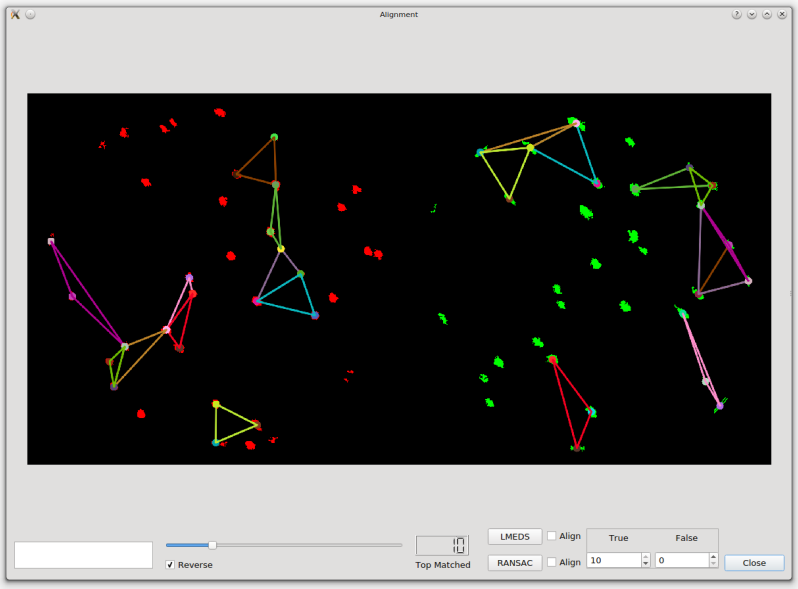


Figure 5-13: K-NT Matching GUI Reverse

After choosing the matching algorithm and clicking "Match" button the user will be promoted with a comparing dialog Figure 5-13, by toggling the slider the user can increase or decrease the number of points collected. The collected points are based on the best matches Figure 5-12 or the worst if the "Reverse" checkbox was checked Figure 5-13.

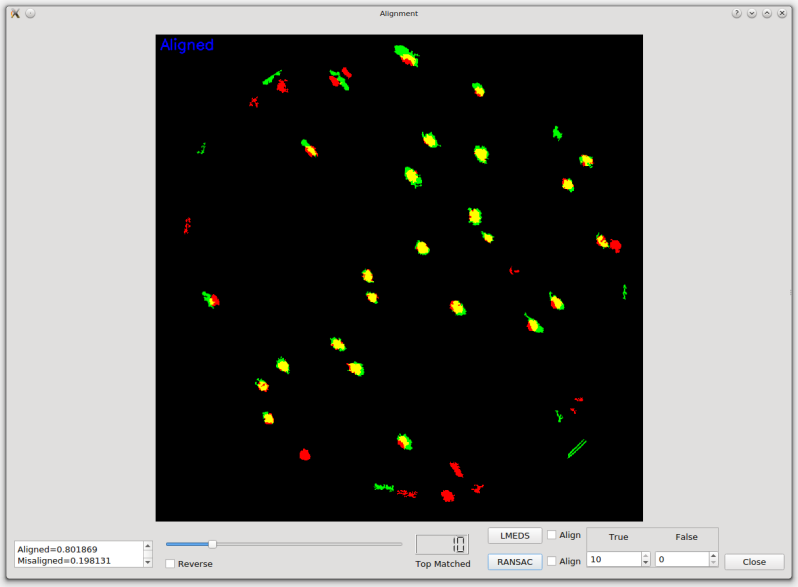


Table 5.1: RANSAC Matching GUI Aligned

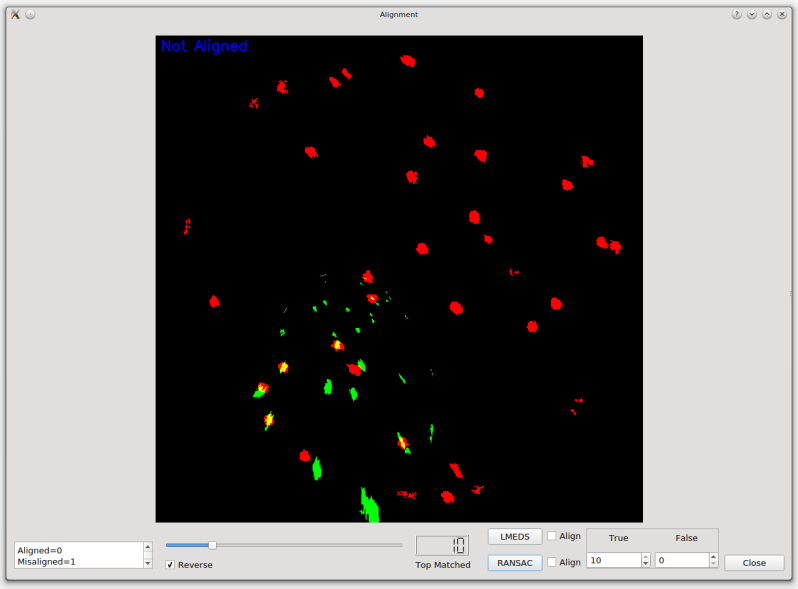


Table 5.2: RANSAC Matching GUI Misaligned

After the user has selected the wanted number of matching points using the slider, the user can click either RANSAC or LMEDS algorithms buttons to perform the alignment. If the corresponding points were correctly matched the result will have the word Aligned at the top left Figure 5.1 otherwise it will have the word Not Aligned Figure 5.2. Also in the bottom left corner of the dialog, a percentage of how will the classifier is performing which will display the weight calculated in the AdaBoost classifier.

Chapter 6

Conclusion and Future Work

6.0.1 Summary of Contributions

We proposed a modified triangulation approach based on Groth [2] and Valdes [3] algorithms to tackle the specific problem of matching cell colonies in pairs of images from Petri dishes. The results were clearly improved by more than 3 times when compared with the existing triangulation matching algorithms which also resulted in an increase in correctly aligning images using RANSAC estimator algorithm by 51.6% compared to Valdes's [3] and 65% compared to Groth's [2]. Experiments showed that due to the nature of the images, the results using SIFT and SURF were not accurate enough to be used in the scope of this application. The matching algorithms are generally slow since they are using nested loops to find the best match so to boost the speed of the algorithms OpenMP library was used to make use of the CPU multi-threading feature.

A new AdaBoost classifier was also introduced and was able to achieve the accuracy of 98.5% at classifying the alignment of images to either correctly aligned or misaligned.

The fusion of algorithms will allow microbiologist to automatically detect and extract Petri dishes from an image and they can easily use the filters to remove unwanted blobs. After applying the filtering, the blobs' alignment process is also automated and the classifier will display the percentage of alignment accuracy to

make the microbiologist aware if there is a possible misalignment. Given a Petri dish containing hundreds of cell colonies, a microbiologist would typically take 15 minutes to perform the matching procedure. On the other hand, the new proposed system could solve it in a matter of seconds.

6.0.2 Future Work

Future work includes the improvement of the filtering approach which can lead to the complete automation of the pairing process, so biology users can receive a complete analysis (e.g., the number of extra cell colonies in one image or another, a count of the total number of colonies, etc) of the final aligned images.

OpenMP library is a powerful tool to optimise code but to really boost the algorithm speed we can use any optimisation used to enhance the speed of the KNN and apply it to K-NT since they have the same concept. Further more, optimising the proposed matching algorithm by using a binary search trees instead of the linear search will speed up the process and it will make it applicable to run on low-powered mobile devices. Optimising the algorithm can also open the doors for database search integration where the user look for a cell colony match in a big database and if there is no match the pattern will be register as a new entry to the database for later searches.

K-NT can be extended to help in building a robust image stitching applications if the right feature extractor where chosen and the end result of the stitched image can be validated using a trained AdaBoost classifier.

Exploring the work done in Preliminary Experiment especially the intersection approach found in Section 3.4.1.2 can be applied in different areas such as face detection since it might works quite well with images that has less missing features.

Bibliography

[1] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

[2] E. J. Groth, "A pattern-matching algorithm for two-dimensional coordinate lists," *The Astronomical Journal*, vol. 91, no. 5, pp. 1244–1248, May 1986.

[3] F. G. Valdes, L. E. Campusano, J. D. Velasquez, and P. B. & Stetson, "Focas automatic catalog matching algorithms," *Publications of the Astronomical Society of the Pacific*, vol. 107, pp. 1119–1128, 1995.

[4] Y. Azzam, K. Kosuge, Z. Wang, A. Alawy, and Y. Hirata, "Telescope automatic alignment and pointing using pattern matching," *The international conference on advanced mechatronics: toward evolutionary fusion of IT and mechatronics: ICAM: abstracts*, vol. 2004, no. 4, 2004.

[5] B. B. Spratling IV and D. Mortari, "A survey on star identification algorithms," *Algorithms*, vol. 2, pp. 93–107, 2009.

[6] M. Marszalek and P. Rokita, "Pattern matching with differential voting and median transformation derivation," in *Computer Vision and Graphics*, 2006, pp. 1002–1007.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[8] S. Alqahtani, A. Barczak, N. Reyes, T. Susnjak, and A. Ganley, "Automatic alignment and comparison on images of petri dishes containing cell colonies," in *Proceedings of the 30th International Conference on Image and Vision Computing New Zealand, IVCNZ 2015, Auckland, New Zealand, November 23-24, 2015*, 2015, pp. –.

[9] J. Lederberg and E. M. Lederberg, "Replica plating and indirect selection of bacterial mutants," *Journal of Bacteriology*, vol. 63, no. 3, p. 399, 1952.

[10] J. Belzer, A. G. Holzman, and A. Kent, *Encyclopedia of Computer Science and Technology: Volume 9-Generative Epistemology of Problem Solving to Laplace and Geometric Transforms*. CRC Press, 1978, vol. 9.

[11] A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., *Computer Vision - ECCV 2002, 7th European Conference on Computer Vision, Copenhagen, Denmark, May 28-31, 2002, Proceedings, Part III*, ser. Lecture Notes in Computer Science, vol. 2352. Springer, 2002.

[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[13] J. Dalal and S. Patel, *Instant OpenCV Starter*. Packt Publishing Ltd, 2013.

[14] K. Kornyakov, "Opencv 3.0 latest news and the roadmap," http://www.slideshare.net/EugeneKhvedchenya/opencv-30-latest-news-and-the-roadmap, 2013.

[15] Jepson and D. Fleet, "Image segmentation," 2007. [Online]. Available: http://www.cs.toronto.edu/ jepson/csc2503/segmentation.pdf

[16] K. K. Singh and A. Singh, "A study of image segmentation algorithms for different types of images," *International Journal of Computer Science*, vol. 7, no. 5, pp. 414–417, 2010.

[17] Y. Saraf, "Algorithms for image segmentation," Ph.D. dissertation, Birla Institute of Technology and Science, Pilani, 2006.

[18] S. Zhu, X. Xia, Q. Zhang, and K. Belloulata, "An image segmentation algorithm in image processing based on threshold segmentation," in *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*. IEEE, 2007, pp. 673–678.

[19] C. Pantofaru and M. Hebert, "A comparison of image segmentation algorithms," *Robotics Institute*, p. 336, 2005.

[20] F. J. Estrada and A. D. Jepson, "Benchmarking image segmentation algorithms," *International Journal of Computer Vision*, vol. 85, no. 2, pp. 167–181, 2009.

[21] J. Van Cleynenbreugel, D. Kratka, L. Berben, M.-H. Smet, G. Marchal, and P. Suetens, "A semiautomatic three-dimensional segmentation method for disarticulation of bone structures on spiral computed tomography images," *Journal of digital imaging*, vol. 8, no. 4, pp. 156–161, 1995.

[22] P. Suetens, E. Bellon, D. Vandermeulen, M. Smet, G. Marchal, J. Nuyts, and L. Mortelmans, "Image segmentation: methods and applications in diagnostic radiology and nuclear medicine," *European journal of radiology*, vol. 17, no. 1, pp. 14–21, 1993.

[23] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[24] M. Lu, "Fast implementation of scale invariant feature transform based on cuda," *Appl. Math*, vol. 7, no. 2L, pp. 717–722, 2013.

[25] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[26] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008.

[27] W. Zhao, X. Chen, J. Cheng, and L. Jiang, "An application of scale-invariant feature transform in iris recognition," in *Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on*. IEEE, 2013, pp. 219–222.

[28] Y. Meng, B. Tiddeman *et al.*, "Implementing the scale invariant feature transform (sift) method," *Citeseer.–2008. : http://www. cs. st-andrews. ac. uk/˜ yumeng/yumeng-SIFTreport-5.18 _bpt. pdf (10.06. 2012)*, 2008.

[29] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[30] J. T. Pedersen, "Study group surf: Feature detection and description," *Department of Computer Science, Aarhus University*, 2011.

[31] B. Pinto and P. Anurenjan, "Video stabilization using speeded up robust features," in *Communications and Signal Processing (ICCSP), 2011 International Conference on*. IEEE, 2011, pp. 527–531.

[32] Z. Zhang, C. Cao, R. Zhang, and J. Zou, "Video copy detection based on speeded up robust features and locality sensitive hashing," in *Automation and Logistics (ICAL), 2010 IEEE International Conference on*. IEEE, 2010, pp. 13–18.

[33] B. Sheta, M. Elhabiby, and N. El-Sheimy, "Assessments of different speeded up robust features (surf) algorithm," 2012.

[34] R. Chaudhry and Y. Ivanov, "Fast approximate nearest neighbor methods for non-euclidean manifolds with applications to human activity analysis in videos," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 735–748.

[35] M. R. Casey, "Fast approximate nearest neighbors," Ph.D. dissertation, Lehigh University, 2006.

[36] M. M. Esmaeili, R. K. Ward, and M. Fatourechi, "A fast approximate nearest neighbor search algorithm in the hamming space," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 12, pp. 2481–2488, 2012.

[37] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, 2009.

[38] Z. Arzoumanian, J. Holmberg, and B. Norman, "An astronomical pattern-matching algorithm for computer-aided identification of whale sharks rhincodon typus," *Journal of Applied Ecology*, vol. 42, no. 6, 2005.

[39] P. B. Stetson, in *Advanced School of Astrophysics, Image and Data Processing/Interstellar Dust.*, B. Barbury, E. Janot-Pacheco, A. M. Magalhães, and S. M. Viegas, Eds. São Paulo, Brazil.: Instituto Astrnomico e Geofsico, 1989.

[40] R. E. Schapire, "Explaining adaboost," in *Empirical inference*. Springer, 2013, pp. 37–52.

[41] T. Jaakkola, "Machine learning : lecture 13," pp. 30–34, 2002. [Online]. Available: http://cs.nyu.edu/ dsontag/courses/ml12/slides/lecture13.pdf

[42] W. Fan, S. J. Stolfo, and J. Zhang, "The application of adaboost for distributed, scalable and on-line learning," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 362–366.

[43] B. Markoski, Z. Ivanković, L. Ratgeber, P. Pecev, and D. Glušac, "Application of adaboost algorithm in basketball player detection," *Acta Polytechnica Hungarica*, vol. 12, no. 1, 2015.

[44] N. Cai, F. Jin, Q. Pan, S.-q. Xu, and F. Li, "Image restoration based on an adaboost algorithm," in *Communications and Information Processing*. Springer, 2012, pp. 294–301.

[45] X. Chen and A. Yuille, "Adaboost learning for detecting and reading text in city scenes," *Department of Statistics, UCLA*, 2011.

[46] M. G. Krishna and A. Srinivasulu, "Face detection system on adaboost algorithm using haar classifiers," *International Journal of Modern Engineering Research*, vol. 2, no. 5, pp. 3556–3560, 2012.

[47] G. Bradski, "Opencv: Examples of use and new applications in stereo, recognition and tracking," in *Proc. Intern. Conf. on Vision Interface*, 2002.

[48] N. Otsu, "A threshold selection method from gray-level histograms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 9, no. 1, pp. 62–66, Jan 1979.

[49] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[50] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer graphics and image processing*, vol. 1, no. 3, pp. 244–256, 1972.

[51] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[52] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[53] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[54] P. J. Rousseeuw, "Least median of squares regression," *Journal of the American statistical association*, vol. 79, no. 388, pp. 871–880, 1984.

[55] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

Appendix A

Tables

The tables in this appendix are some of the dataset that was collected during this research. Table A.1 is part of the final result gathered by testing the 60 pairs of Petri dishes. Tables A.2, A.3 and A.4 are some of the early testing results in the matching algorithms stage. Table A.5 shows the attributes extracted from the alignment stage and was used in the preliminary experiments to generate a classifier. On the other hand, Table A.6 is the attributes that was fed into AdaBoost training stage and resulted in creating the AdaBoost classifier.

Method	TRUE	FALSE	LMEDS	RANSAC	Method	TRUE	FALSE	LMEDS	RANSAC	Method	TRUE	FALSE	LMEDS	RANSAC
Groth	10	0	1	1	**Valdes**	8	2	0	1	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	10	0	1	1	**K-NT**	10	0	1	1
Groth	5	5	0	1	**Valdes**	2	8	0	1	**K-NT**	5	5	1	1
Groth	7	3	1	1	**Valdes**	6	4	1	1	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	0	10	0	0	**K-NT**	9	1	1	1
Groth	0	10	0	0	**Valdes**	0	10	0	0	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	0	10	0	0	**K-NT**	7	3	1	1
Groth	2	8	0	0	**Valdes**	0	10	0	0	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	1	9	0	0	**K-NT**	8	2	1	1
Groth	1	9	0	0	**Valdes**	1	9	0	0	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	1	9	0	0	**K-NT**	10	0	1	1
Groth	2	8	0	1	**Valdes**	4	6	0	1	**K-NT**	10	0	1	1
Groth	4	6	0	1	**Valdes**	7	3	1	1	**K-NT**	5	5	1	1
Groth	10	0	1	1	**Valdes**	8	2	1	1	**K-NT**	8	2	1	1
Groth	3	7	0	1	**Valdes**	0	10	0	0	**K-NT**	9	1	1	1
Groth	0	10	0	0	**Valdes**	1	9	0	0	**K-NT**	4	6	0	1
Groth	0	10	0	0	**Valdes**	0	10	0	0	**K-NT**	7	3	1	1
Groth	6	4	1	1	**Valdes**	5	5	1	1	**K-NT**	9	1	0	1
Groth	2	8	0	0	**Valdes**	6	4	1	0	**K-NT**	7	3	1	1
Groth	2	8	0	0	**Valdes**	2	8	0	1	**K-NT**	10	0	1	1
Groth	1	9	0	0	**Valdes**	10	0	0	1	**K-NT**	10	0	1	1
Groth	1	9	0	0	**Valdes**	0	10	0	0	**K-NT**	10	0	1	1
Groth	2	8	0	1	**Valdes**	2	8	0	1	**K-NT**	10	0	1	1
Groth	6	4	1	1	**Valdes**	2	8	0	1	**K-NT**	10	0	1	1
Groth	4	6	0	1	**Valdes**	4	6	0	1	**K-NT**	10	0	1	1
Groth	0	10	0	0	**Valdes**	2	8	0	1	**K-NT**	7	3	1	1
Groth	1	9	0	0	**Valdes**	5	5	1	1	**K-NT**	10	0	1	1
Groth	2	8	0	0	**Valdes**	1	9	0	0	**K-NT**	10	0	1	1
Groth	1	9	0	0	**Valdes**	2	8	0	1	**K-NT**	10	0	1	1
Groth	1	9	0	0	**Valdes**	0	10	0	0	**K-NT**	10	0	0	1
Groth	1	9	0	0	**Valdes**	0	10	0	0	**K-NT**	5	5	1	1
Groth	0	10	0	0	**Valdes**	1	9	0	0	**K-NT**	6	4	0	1
Groth	1	9	0	0	**Valdes**	3	7	0	1	**K-NT**	5	5	1	1
Groth	0	10	0	0	**Valdes**	2	8	0	0	**K-NT**	6	4	1	0
Groth	0	10	0	0	**Valdes**	1	9	0	0	**K-NT**	5	5	1	1
...	...	...	...	...	**...**	...	...	...	...	**...**	...	...	...	...
Total	**121**	**479**	**7**	**20**		**146**	**454**	**10**	**28**		**477**	**123**	**49**	**59**

Table A.1: Triangulation Based Matching Algorithms Performance Summery

Methods		Pixel Matching		Correctly	Aligned	
Detecting	**Matching**	**FALSE**	**TRUE**	**Matched Point**	**RANSAC**	**LMEDS**
Groth	Manual	21	9	9	TRUE	FALSE
Valdes	Manual	21	9	6	TRUE	FALSE
K-NT n=8	Manual	0	30	8	TRUE	TRUE
K-NT n=5	Manual	0	30	9	TRUE	TRUE
Groth	SURF	27	3	3	FALSE	FALSE
Valdes	SURF	27	3	3	FALSE	FALSE
K-NT n=8	SURF	30	0	0	FALSE	FALSE
K-NT n=5	SURF	21	9	5	TRUE	FALSE
FLANN	SURF	30	0	0	FALSE	FALSE
BFM	SURF	30	0	0	FALSE	FALSE
Groth	SIFT	30	0	0	FALSE	FALSE
Valdes	SIFT	30	0	0	FALSE	FALSE
K-NT n=8	SIFT	12	18	5	TRUE	FALSE
K-NT n=5	SIFT	21	9	3	FALSE	FALSE
FLANN	SIFT	30	0	0	FALSE	FALSE
BFM	SIFT	0	0	0	FALSE	FALSE

Table A.2: Figure B-1 Results

Methods		Pixel Matching		Correctly	Aligned	
Detecting	**Matching**	**FALSE**	**TRUE**	**Matched Point**	**RANSAC**	**LMEDS**
Groth	Manual	15	15	13	TRUE	TRUE
Valdes	Manual	6	27	16	TRUE	TRUE
K-NT n=8	Manual	0	30	11	TRUE	TRUE
K-NT n=5	Manual	0	30	12	TRUE	TRUE
Groth	SURF	30	0	0	FALSE	FALSE
Valdes	SURF	27	3	3	FALSE	FALSE
K-NT n=8	SURF	6	24	10	TRUE	TRUE
K-NT n=5	SURF	0	30	13	TRUE	TRUE
FLANN	SURF	30	0	0	FALSE	FALSE
BFM	SURF	30	0	0	FALSE	FALSE
Groth	SIFT	30	0	0	FALSE	FALSE
Valdes	SIFT	30	0	0	FALSE	FALSE
K-NT n=8	SIFT	30	0	0	FALSE	FALSE
K-NT n=5	SIFT	30	0	0	FALSE	FALSE
FLANN	SIFT	30	0	0	FALSE	FALSE
BFM	SIFT	30	0	0	FALSE	FALSE

Table A.3: Figure B-2 Results

Methods		Pixel Matching		Correctly	Aligned	
Detecting	**Matching**	**FALSE**	**TRUE**	**Matched Point**	**RANSAC**	**LMEDS**
Groth	Manual	30	0	0	FALSE	FALSE
Valdes	Manual	24	6	6	TRUE	FALSE
K-NT n=8	Manual	6	24	11	TRUE	TRUE
K-NT n=5	Manual	9	21	11	TRUE	TRUE
Groth	SURF	30	0	0	FALSE	FALSE
Valdes	SURF	30	0	0	FALSE	FALSE
K-NT n=8	SURF	30	0	0	FALSE	FALSE
K-NT n=5	SURF	30	0	0	FALSE	FALSE
FLANN	SURF	30	0	0	FALSE	FALSE
BFM	SURF	30	0	0	FALSE	FALSE
Groth	SIFT	30	0	0	FALSE	FALSE
Valdes	SIFT	30	0	0	FALSE	FALSE
K-NT n=8	SIFT	30	0	0	FALSE	FALSE
K-NT n=5	SIFT	30	0	0	FALSE	FALSE
FLANN	SIFT	26	4	4	FALSE	FALSE
BFM	SIFT	30	0	0	FALSE	FALSE

Table A.4: Figure B-3 Results

is_aligned	red_count	red_pixels	red_count_after	red_pixels_after	green_count	green_pixels	green_count_after	green_pixels_after	yellow_pixels	yellow_count
A	18	3508	34	1298	12	3974	45	2054	2054	12
A	18	3508	37	1216	12	3962	49	1967	2124	11
A	18	3508	37	1255	12	3883	57	1924	2083	13
A	18	3508	39	1242	12	3876	54	1906	2093	13
A	18	3508	44	1153	12	3907	57	1839	2199	11
B	18	3508	34	1827	12	3995	46	2563	1533	12
B	18	3508	27	2589	12	4582	32	3756	857	5
A	18	3508	46	1195	12	3910	60	1887	2153	12
A	18	3508	41	1202	12	3928	56	1892	2157	13
A	18	3508	41	1173	12	3924	59	1873	2182	12
A	18	3508	45	1175	12	3890	60	1837	2191	11
A	18	3508	35	1261	12	3957	45	2006	2071	11
A	18	3508	32	1238	12	3951	49	1958	2118	11
A	18	3508	44	1166	12	3923	57	1861	2191	11
A	18	3508	36	887	14	4435	63	1995	2519	10
A	18	3508	30	889	14	4419	64	1983	2523	10
A	18	3508	33	880	14	4425	59	1988	2530	10
A	18	3508	35	905	14	4428	62	2012	2512	10
A	18	3508	34	879	14	4435	64	1985	2527	10
A	18	3508	35	876	14	4420	64	1969	2537	10
A	18	3508	34	878	14	4421	64	1967	2533	10
B	18	3508	26	2579	14	4350	35	3528	875	6
B	18	3508	28	2766	14	3177	29	2531	671	5
A	18	3508	33	919	14	4405	60	2024	2473	10
A	18	3508	32	952	14	4388	56	2043	2448	10
A	18	3508	36	901	14	4426	60	2000	2517	10
A	18	3508	36	868	14	4413	58	1954	2543	10
A	18	3508	36	885	14	4428	60	1992	2528	10
A	8	1414	25	269	15	3827	42	2407	1152	6
B	8	1460	25	472	15	3827	34	2592	993	6
A	8	1419	25	268	15	3827	42	2416	1162	6
A	8	1419	24	266	15	3827	39	2406	1161	6
A	8	1446	26	381	15	3827	37	2493	1073	6
A	8	1418	25	316	15	3827	40	2455	1107	6
A	8	1416	26	264	15	3827	40	2409	1159	6
A	8	1419	24	266	15	3827	39	2406	1161	6
A	8	1498	19	815	8	1487	13	674	684	4
A	8	1512	23	826	8	1487	12	673	689	4
B	8	498	70	498	8	1487	8	1486	0	0
A	8	1458	25	775	8	1487	13	683	685	4
A	8	1497	22	858	8	1487	13	743	641	4
B	8	1368	19	839	8	1487	12	869	531	3
A	8	1498	19	815	8	1487	13	674	684	4
A	8	1454	25	775	8	1487	15	688	685	4
B	8	498	70	498	8	1487	8	1486	0	0
B	8	1783	38	1738	8	1487	9	1434	45	1
A	8	1458	25	775	8	1487	13	683	685	4
A	26	7644	70	3242	16	4852	76	1080	4039	14
A	26	7644	64	3357	16	4905	78	1220	3940	13
B	26	7644	50	6661	16	7281	41	6532	832	10
A	26	7644	72	3245	16	4858	92	1069	4062	14
A	26	7644	69	3271	16	4862	93	1084	4026	13
A	26	7644	75	3239	16	4859	84	1075	4059	12
A	26	7644	72	3256	16	4868	83	1078	4043	13
A	26	7644	68	3212	16	4842	86	1020	4091	13
A	26	7644	72	3273	16	4839	90	1095	4023	11
A	26	7644	80	3196	16	4822	89	1018	4097	12
B	26	7644	30	7045	16	3462	26	2942	549	7
B	26	7644	29	7102	16	3070	29	2707	436	6
B	26	7644	25	7058	16	6750	24	6175	582	10
...	...	...	...	...	...	...	...	...	...	...

Table A.5: Original Attributes

is_aligned	green_yellow	green_yellow_after	red_yellow	red_yellow_after	red_green_yellow	red_green_yellow_after	red_yellow_count	green_yellow_count	red_yellow_count_after	green_yellow_count_after
A	0.5168595873	1	0.5855188141	1.5824345146	0.5490510559	1.2255369928	0.6666666667	1	0.3529411765	0.2666666667
A	0.5360928824	1.0798169802	0.6054732041	1.7467105263	0.5686746988	1.3345900094	0.6111111111	0.9166666667	0.2972972973	0.2244897959
A	0.5364408962	1.0826403326	0.5937856328	1.6597609562	0.5636585036	1.3104749921	0.7222222222	1.0833333333	0.3513513514	0.2280701754
A	0.5399896801	1.0981112277	0.59663626	1.6851851852	0.5669014085	1.3297331639	0.7222222222	1.0833333333	0.3333333333	0.2407407407
A	0.5628359355	1.1957585644	0.6268529076	1.9071986123	0.5931220499	1.4699197861	0.6111111111	0.9166666667	0.25	0.1929824561
B	0.3837296621	0.5981271947	0.4370011403	0.8390804598	0.4086365454	0.698405467	0.6666666667	1	0.3529411765	0.2608695652
B	0.1870362287	0.2281682641	0.2442987457	0.3310158362	0.2118665019	0.2701339638	0.2777777778	0.4166666667	0.1851851852	0.15625
A	0.5506393862	1.1409644939	0.6137400228	1.8016736402	0.5804799137	1.3971447112	0.6666666667	1	0.2608695652	0.2
A	0.5491344196	1.1400634249	0.6148802737	1.7945091514	0.5801506186	1.3943115708	0.7222222222	1.0833333333	0.3170731707	0.2321428571
A	0.5560652396	1.1649759744	0.6220068415	1.8601875533	0.5871905274	1.4326986211	0.6666666667	1	0.2926829268	0.2033898305
A	0.5632390746	1.1927054981	0.6245724059	1.8646808511	0.5923222493	1.4548472776	0.6111111111	0.9166666667	0.2444444444	0.1833333333
A	0.5233762952	1.0324027916	0.5903648803	1.6423473434	0.5548559946	1.2678298133	0.6111111111	0.9166666667	0.3142857143	0.2444444444
A	0.5360668185	1.0817160368	0.6037628278	1.7108239095	0.5679045448	1.3254067584	0.6111111111	0.9166666667	0.34375	0.2244897959
A	0.5585011471	1.1773240193	0.6245724059	1.8790737564	0.5896918315	1.4476379253	0.6111111111	0.9166666667	0.25	0.1929824561
A	0.5679819617	1.2626566416	0.7180729761	2.8399098083	0.6342691678	1.7480916031	0.5555555556	0.7142857143	0.2777777778	0.1587301587
A	0.5709436524	1.2723146747	0.7192132269	2.8380202475	0.6365585972	1.7569637883	0.5555555556	0.7142857143	0.3333333333	0.15625
A	0.5717514124	1.2726358149	0.7212086659	2.875	0.6378419261	1.7642956764	0.5555555556	0.7142857143	0.303030303	0.1694915254
A	0.5672990063	1.2485089463	0.7160775371	2.7756906077	0.6330645161	1.7223174494	0.5555555556	0.7142857143	0.2857142857	0.1612903226
A	0.5697857948	1.2730478589	0.7203534778	2.8748577929	0.6362835201	1.7646648045	0.5555555556	0.7142857143	0.2941176471	0.15625
A	0.5739819005	1.2884713052	0.7232041049	2.8961187215	0.6400100908	1.7834797891	0.5555555556	0.7142857143	0.2857142857	0.15625
A	0.572947297	1.2877478393	0.722063854	2.8849658314	0.6389204187	1.7806678383	0.5555555556	0.7142857143	0.2941176471	0.15625
B	0.2011494253	0.248015873	0.2494298746	0.3392787902	0.2227029779	0.2865564107	0.3333333333	0.4285714286	0.2307692308	0.1714285714
B	0.2112055398	0.2651126037	0.191277081	0.2425885756	0.2007479432	0.2533509534	0.2777777778	0.3571428571	0.1785714286	0.1724137931
A	0.5614074915	1.2218379447	0.7049600912	2.690968444	0.6250473904	1.6805980292	0.5555555556	0.7142857143	0.303030303	0.1666666667
A	0.5578851413	1.1982378855	0.6978335234	2.5714285714	0.6200607903	1.6347245409	0.5555555556	0.7142857143	0.3125	0.1785714286
A	0.5686850429	1.2585	0.7175028506	2.7935627081	0.6344844971	1.7352637022	0.5555555556	0.7142857143	0.2777777778	0.1666666667
A	0.5762519828	1.301432958	0.7249144812	2.9297235023	0.6420906451	1.8022678951	0.5555555556	0.7142857143	0.2777777778	0.1724137931
A	0.5709123758	1.2690763052	0.7206385405	2.8564971751	0.6370967742	1.7573861661	0.5555555556	0.7142857143	0.2777777778	0.1666666667
A	0.301019075	0.4786040715	0.8147100424	4.282527881	0.4396107613	0.8609865471	0.75	0.4	0.24	0.1428571429
B	0.2594721714	0.3831018519	0.6801369863	2.1038135593	0.3756383582	0.6481723238	0.75	0.4	0.24	0.1764705882
A	0.3036320878	0.4809602649	0.8188865398	4.3358208955	0.4430041937	0.8658718331	0.75	0.4	0.24	0.1428571429
A	0.3033707865	0.4825436409	0.8181818182	4.3646616541	0.4426229508	0.869011976	0.75	0.4	0.25	0.1538461538
A	0.2803762738	0.4304051344	0.7420470263	2.8162729659	0.4069789494	0.7466945024	0.75	0.4	0.2307692308	0.1621621622
A	0.2892605174	0.4509164969	0.7806770099	3.503164557	0.4221163012	0.7989895345	0.75	0.4	0.24	0.15
A	0.302848184	0.4811124948	0.8185028249	4.3901515152	0.4421132939	0.8671904227	0.75	0.4	0.2307692308	0.15
A	0.3033707865	0.4825436409	0.8181818182	4.3646616541	0.4426229508	0.869011976	0.75	0.4	0.25	0.1538461538
A	0.4599865501	1.0148367953	0.4566088117	0.8392638037	0.4582914573	0.9187374077	0.5	0.5	0.2105263158	0.3076923077
A	0.4633490249	1.0237741456	0.4556878307	0.8341404358	0.4594864955	0.9192795197	0.5	0.5	0.1739130435	0.3333333333
B	0	0	0	0	0	0	0	0	0	0
A	0.4606590451	1.0029282577	0.4698216735	0.8838709677	0.4651952462	0.9396433471	0.5	0.5	0.16	0.3076923077
A	0.431069267	0.8627187079	0.4281897128	0.7470862471	0.4296246649	0.8007495315	0.5	0.5	0.1818181818	0.3076923077
B	0.3570948218	0.6110471807	0.3881578947	0.6328963051	0.3719789842	0.6217798595	0.375	0.375	0.1578947368	0.25
A	0.4599865501	1.0148367953	0.4566088117	0.8392638037	0.4582914573	0.9187374077	0.5	0.5	0.2105263158	0.3076923077
A	0.4606590451	0.9956395349	0.4711141678	0.8838709677	0.4658279497	0.9364319891	0.5	0.5	0.16	0.2666666667
B	0	0	0	0	0	0	0	0	0	0
B	0.030262273	0.0313807531	0.0252383623	0.0258918297	0.0275229358	0.0283732661	0.125	0.125	0.0263157895	0.1111111111
A	0.4606590451	1.0029282577	0.4698216735	0.8838709677	0.4651952462	0.9396433471	0.5	0.5	0.16	0.3076923077
A	0.8324402308	3.7398148148	0.5283882784	1.2458359038	0.646446863	1.8690421101	0.5384615385	0.875	0.2	0.1842105263
A	0.8032619776	3.2295081967	0.515436944	1.1736669646	0.6279384812	1.7216517369	0.5	0.8125	0.203125	0.1666666667
B	0.1142700179	0.1273729333	0.1088435374	0.1249061702	0.1114907873	0.1261274919	0.3846153846	0.625	0.2	0.243902439
...	...	...	...	...	...	...	...	...	...	...

Table A.6: Normalised Attributes

Appendix B

Figures

This appendix presents three image examples of the real biology data that was collected from the biology research group. All blue Petri dishes are replicas of the master Petri dishes which are beige coloured. The caption of each figure show the cell count category that they belong to for example caption from Figure B-1 "Image t0.100.2" means that the dishes involved are considered in the 100 cells category..

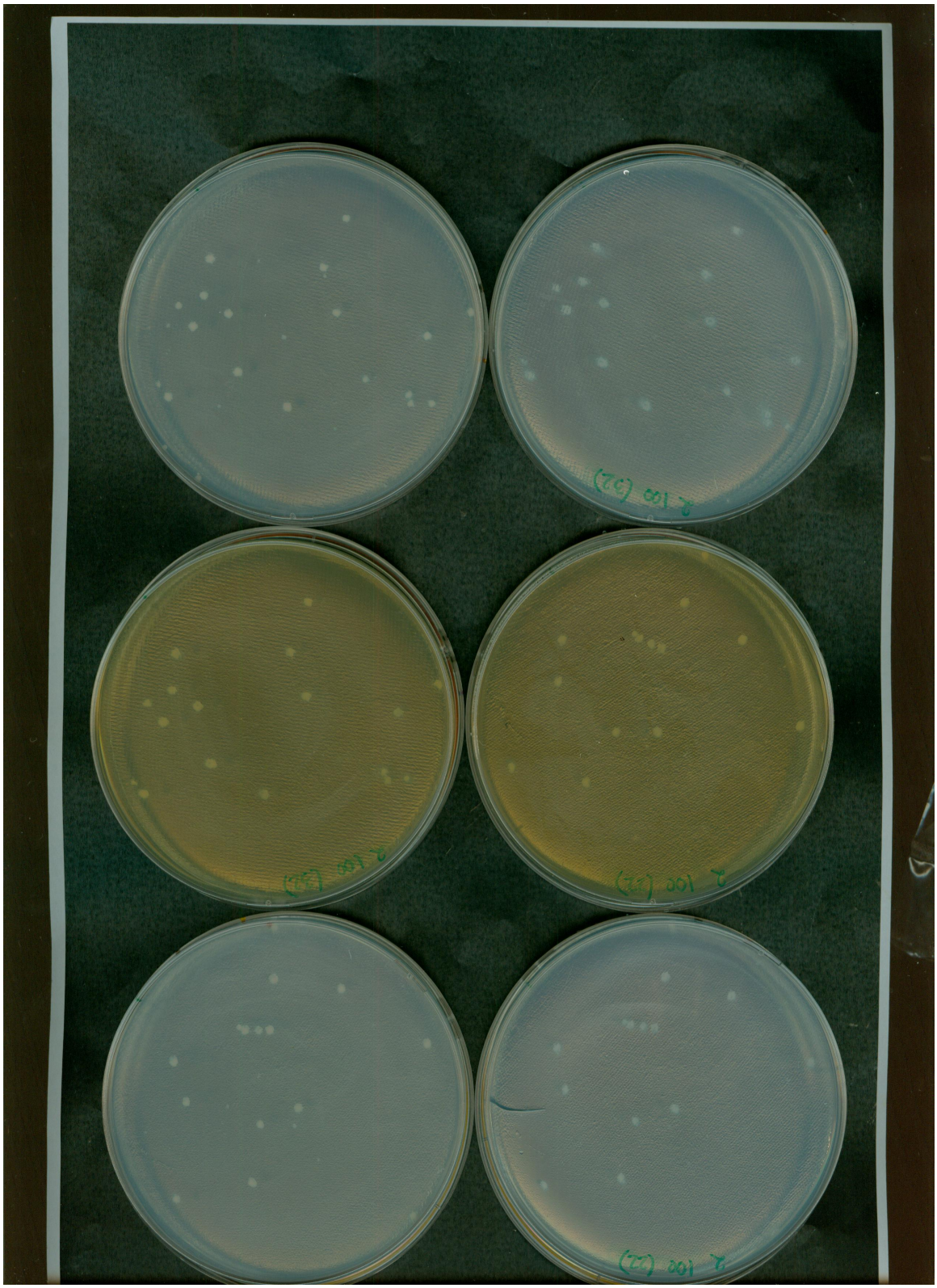


Figure B-1: Image t0.100.2

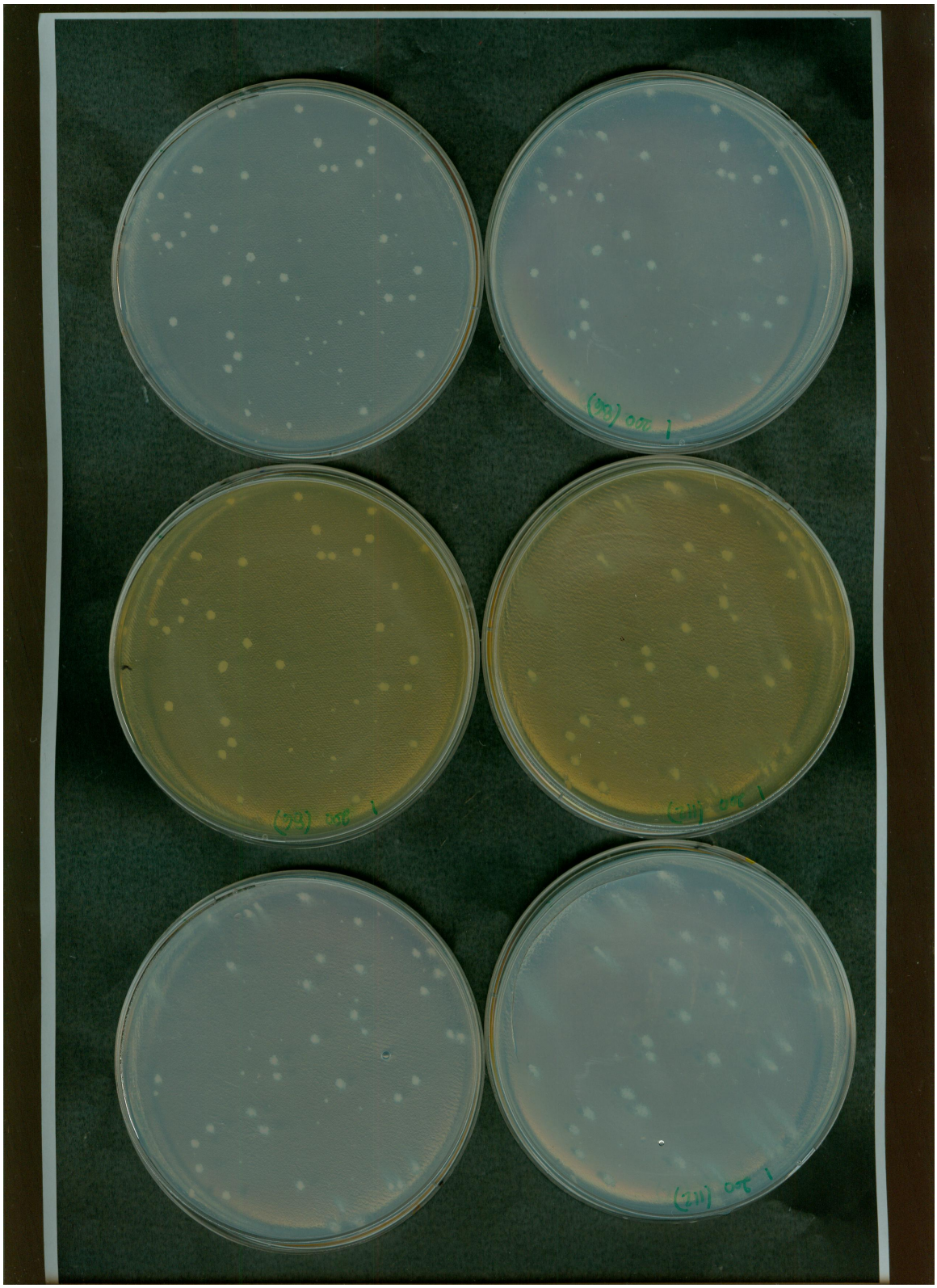


Figure B-2: Image t0.200.1

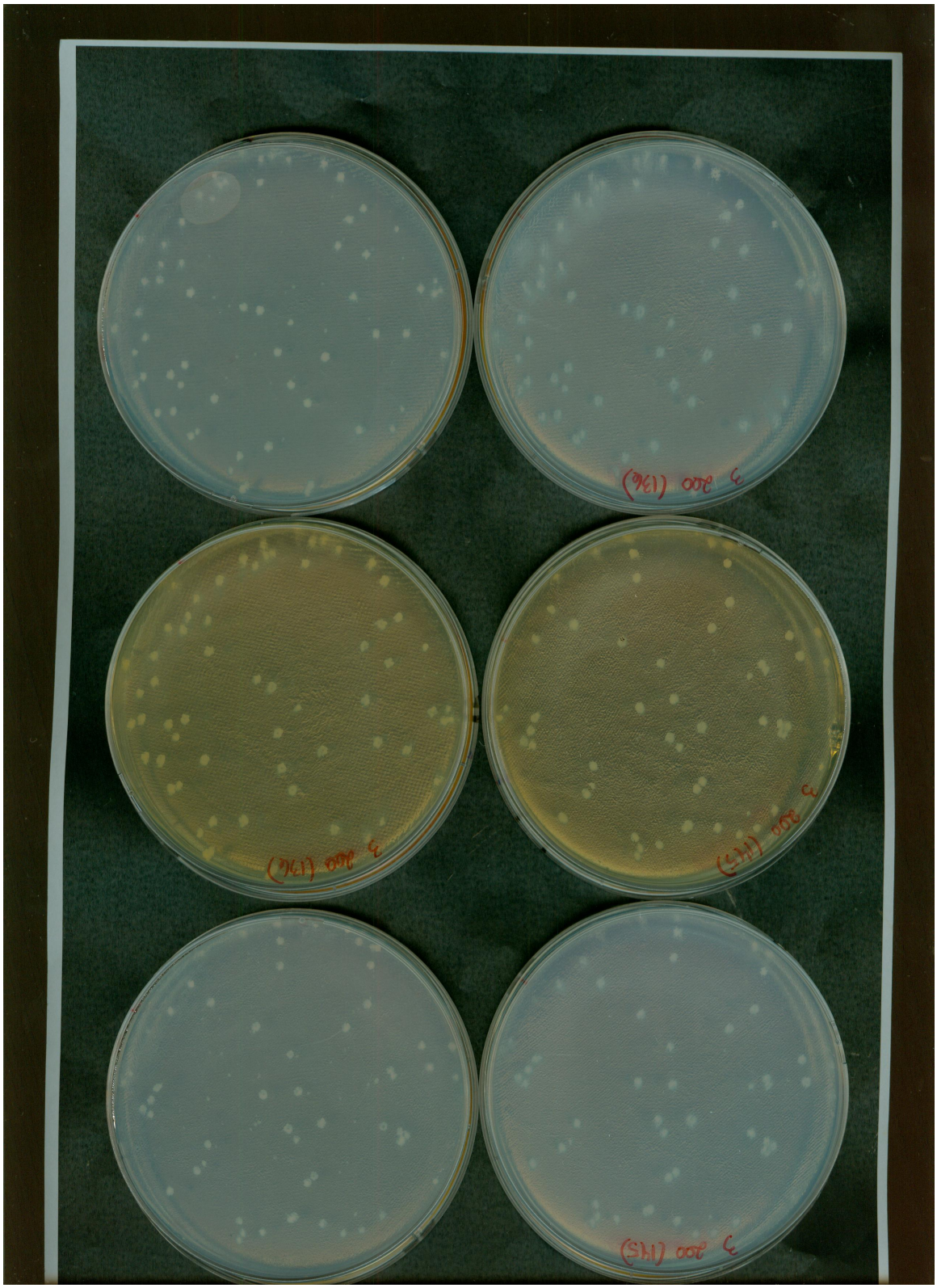


Figure B-3: Image t1.200.3

Appendix C

Code

Listing C.1 is the code of the AdaBoost classifier. Listing C.2 and C.3 are the code of the class that hold all the matching algorithms, namely Groth, Valdes, K-NT, FANN, BFM.

```
1  #ifndef ADABOOST
2  #define ADABOOST
3  #include <string>
4  class AdaBoost{
5  public:
6      ///
7      /// \brief Evaluate the image alignment after getting the count of
           each pixels
8      /// \param green_yellow_count_after: The yellow pixels count / (the
           green pixels count from the aligned image)
9      /// \param green_yellow: The yellow pixels count / the green pixels
           count
10     /// \param red_green_yellow_after:   The yellow pixels count / (the
           red + green pixels count from the aligned image)/2
11     /// \param red_green_yellow:   The yellow pixels count / the red +
           green pixel count
12     /// \param red_yellow_after:   The yellow pixels count / (the red
           pixels count from the aligned image)
13     ///
14     void Evaluate(float green_yellow_count_after, float green_yellow,
```

```
        float red_green_yellow_after, float red_green_yellow, float
        red_yellow_after)
15  {
16      A=M=0;
17      (red_green_yellow > 0.4206722775)                    ?  A+= 4.50
            : M += 4.50;
18      (red_green_yellow> 0.28496798155)                    ?  A+= 4.58
            : M += 4.58;
19      (green_yellow_count_after > 0.48074074074999995)     ?  A+= 1.30
            : M += 1.30;
20      (red_green_yellow_after > 0.7367843402)              ?  A+= 1.18
            : M += 1.18;
21      (green_yellow > 0.24484761465)                       ?  A+= 1.38
            : M += 1.38;
22      (red_yellow_after > 1.2491860691)                    ?  A+= 1.24
            : M += 1.24;
23      (green_yellow_count_after > 0.39811912225)           ?  A+= 1.17
            : M += 1.17;
24      (red_green_yellow_after > 0.7864564795)              ?  A+= 0.70
            : M += 0.70;
25      A /= 16.05;
26      M /= 16.05;
27  }
28  ///
29  /// \brief isAligned is a function that checks if the image is
        aligned or not after performing the Evaluate function.
30  /// \return boolen true for aligned or false for misaligned
31  ///
32  bool isAligned(){
33      return A>M;
34  }
35  ///
36  /// \brief getPercentage return the accuracy of the classification
37  /// \return float percentage
38  ///
39  float getPercentage(){
```

```
40          return A;
41      }
42      ///
43      /// \brief print the percentage(accuracy) of the evaluation
44      /// \return string the percentage
45      ///
46      std::string print(){
47          std::stringstream buffer;
48          buffer<<"Aligned="<<A<<std::endl<<"Misaligned="<<M;
49          return buffer.str();
50      }
51
52  private:
53      float A, M;
54  };
55  #endif // ADABOOST
```

Listing C.1: AdaBoost Class Code

```
1  #ifndef DIALOGCOMPARE_H
2  #define DIALOGCOMPARE_H
3
4  #include <QDialog>
5  #include "mattoqimage.h"
6  #include <iostream>
7  #include <datastructs.h>
8
9
10
11 namespace Ui {
12 class DialogCompare;
13 }
14
15 class DialogCompare : public QDialog
16 {
17     Q_OBJECT
18
```

```
19 public:
20     ///
21     /// \brief DialogCompare the constructor that will setup the
           algorithms used
22     /// \param image_1_src the source of the first image (CV_8UC3)
23     /// \param image_2_src the source of the second image (CV_8UC3)
24     /// \param keypoints_1 the position of cells in the first image
25     /// \param keypoints_2 the position of cells in the second image
26     /// \param desc_1 the first image descriptor if FLANN or BFM to be
           used
27     /// \param desc_2 the second image descriptor if FLANN or BFM to be
           used
28     /// \param alg the mathcing algorithm to be used
29     /// \param e_n Number of neighbours if K-TN(MatchSAFAR) to be used
30     /// \param debug_mode enables debug mode
31     ///
32     DialogCompare(cv::Mat image_1_src, cv::Mat image_2_src,
33                   std::vector<KeyPoint> keypoints_1,
34                   std::vector<KeyPoint> keypoints_2, Mat desc_1, Mat
                       desc_2,
35                   AlgorithmsName::MatcherEnum alg, int e_n, bool
                       debug_mode);
36     void DisplayCompare(std::vector< DMatch > matches,
37                         std::vector<KeyPoint> one,
38                         std::vector<KeyPoint> two);
39     ~DialogCompare();
40
41     ///
42     /// \brief getResults extract all needed information to apply the
           AdaBoost classifier
43     /// \return string that has all pixel counts for each image
44     ///
45     QString getResults();
46 private slots:
47     void on_pushButton_clicked();
48     void moving_slider();
```

```
49      void on_pushButton_ransac_clicked();
50      void on_pushButton_lmeds_clicked();
51
52      void on_spinBox_true_valueChanged(int arg1);
53
54      void on_spinBox_false_valueChanged(int arg1);
55
56  private:
57      std::vector<Point2d> OnlyDotsGr(cv::Mat *src);
58      void MatchAKAZE(cv::Mat image_1, cv::Mat image_2);
59      void MatchFLANN(std::vector<KeyPoint> keypoints_1,
60                      std::vector<KeyPoint> keypoints_2,
61                      Mat descriptors_1, Mat descriptors_2);
62
63      void MatchBFM(std::vector<KeyPoint> keypoints_1, std::vector<
            KeyPoint> keypoints_2, Mat descriptors_1, Mat descriptors_2);
64
65      void MatchGROTH(std::vector<Point2f> mc1, std::vector<Point2f> mc2);
66      ///
67      /// \brief MatchSAFAR applies function FindAllTriangles_SAFAR then
            MatchLoopTriangles_SAFAR to the two sets of points
68      /// \param mc1 cell positions from image 1
69      /// \param mc2 cell positions from image 2
70      ///
71      void MatchSAFAR(std::vector<Point2f> mc1, std::vector<Point2f> mc2);
72      void MatchVALDES(std::vector<Point2f> mc1, std::vector<Point2f> mc2)
            ;
73      void MatchLoopTriangles_GROTH(std::vector<TriangleG> a, std::vector<
            TriangleG> b);
74      ///
75      /// \brief MatchLoopTriangles_SAFAR apply the proposed matching
            algorithm K_NT on the two sets of triangles
76      /// \param a all possiable triangle from image 1 cells positions
77      /// \param b all possiable triangle from image 2 cells positions
78      /// \param fit set false if neighbors number is less than the cell
            count
```

```
79      ///
80      void MatchLoopTriangles_SAFAR(std::vector<TriangleV> a, std::vector<
            TriangleV> b, bool fit);
81      void MatchLoopTriangles_VALDES(std::vector<TriangleV> a, std::vector
            <TriangleV> b);
82
83      std::vector<TriangleG> FindAllTriangles_GROTH(std::vector<Point2f>
            mc1, int x, int y);
84      ///
85      /// \brief FindAllTriangles_SAFAR find all possiable triangles
            combination from a set of points
86      /// \param mc1 set of points to genrate the triangles
87      /// \param fit set false if neighbors number is less than the cell
            count
88      /// \return set of triangles
89      ///
90      std::vector<TriangleV> FindAllTriangles_SAFAR(std::vector<Point2f>
            mc1, bool fit);
91      std::vector<TriangleV> FindAllTriangles_VALDES(std::vector<Point2f>
            mc1);
92      template <class G_TRI>
93      ///
94      /// \brief RateNSliderS is a function used by the user throughout
            the gui to select the top or bottom matched triangle
95      /// to peform the alignment
96      /// \param list_of_tri List of matching triangles
97      /// \param img_1 Image source 1
98      /// \param img_2 Image source 2
99      /// \param top Number of matching triangles to be considered
100     /// \param reverse Reverse the search to get the bottom matches
            instead of the top ones
101     ///
102     void RateNSliderS(std::vector<G_TRI> &list_of_tri, Mat *img_1, Mat *
            img_2, int top, bool reverse);
103     void Alignment(int method);
104     void ToggleAlignment(bool status);
```

```
105
106     Ui::DialogCompare *ui;
107     cv::Mat images_src_[2];
108     std::vector<TriangleG> b_f_g, a_f_g;
109     std::vector<TriangleV> b_f_v, a_f_v;
110     std::vector<Point2d> corros_img1, corros_img2;
111     bool flip, debug_mode;
112     int red_count, green_count;
113     AlgorithmsName::MatcherEnum algorithm;
114     const uint N_SIZE;
115     QString store_them;
116 };
117
118 #endif // DIALOGCOMPARE_H
```

Listing C.2: DialogCompare Header Code

```
1  #include "dialogcompare.h"
2  #include "ui_dialogcompare.h"
3  #include <vector>
4  #include <iostream>
5  #include <opencv2/features2d.hpp>
6  #include <opencv2/imgproc.hpp>
7  #include <opencv2/xfeatures2d.hpp>
8  #include <opencv2/core/core.hpp>
9  #include <opencv2/imgcodecs.hpp>
10 #include <opencv2/highgui/highgui.hpp>
11 #include <opencv2/calib3d.hpp>
12 #include <timer.h>
13 #include <omp.h>
14 #include <adaboost.h>
15 double difangrad(double x, double y) {
16
17     double a = x - y;
18     if(a < -180) a += 360;
19     if(a > 180) a -= 360;
20     return a;
```

```
21 }
22
23
24 DialogCompare::DialogCompare(Mat image_1_src, Mat image_2_src,
25                              std::vector<KeyPoint> keypoints_1,
26                              std::vector<KeyPoint> keypoints_2,
27                              Mat desc_1, Mat desc_2,
28                              AlgorithmsName::MatcherEnum alg, int e_n,
29                              bool debug) :
30     ui(new Ui::DialogCompare), N_SIZE(e_n)
31 {
32     flip = false;
33     debug_mode = debug;
34     ui->setupUi(this);
35     if(image_1_src.data && image_2_src.data){
36         connect(ui->horizontalSlider_rate, SIGNAL(valueChanged(int)),
              this, SLOT(moving_slider()));
37         algorithm = alg;
38         images_src_[0] = image_1_src;
39         images_src_[1] = image_2_src;
40         std::vector<Point2f> mc1, mc2;
41         KeyPoint::convert(keypoints_1, mc1);
42         KeyPoint::convert(keypoints_2, mc2);
43         red_count   = keypoints_2.size();
44         green_count = keypoints_1.size();
45         Timer time;
46         time.Start();
47         switch (alg) {
48         case AlgorithmsName::alg_FLANN:
49             MatchFLANN(keypoints_1, keypoints_2, desc_1, desc_2);
50             break;
51         case AlgorithmsName::alg_BFM:
52             MatchBFM(keypoints_1, keypoints_2, desc_1, desc_2);
53             break;
54         case AlgorithmsName::alg_GROTH:
55             MatchGROTH(mc1, mc2);
```

```
56                  break;
57              case AlgorithmsName::alg_VALDES:
58                  MatchVALDES(mc1, mc2);
59                  break;
60              case AlgorithmsName::alg_SAFAR:
61                  MatchSAFAR(mc1, mc2);
62                  break;
63              default:
64                  break;
65              }
66              time.End();
67          }
68  }
69  QString DialogCompare::getResults(){
70      return  (ui->checkBox_ransac->isChecked()?"A,":"B,")+store_them;
71  }
72
73  void DialogCompare::ToggleAlignment(bool status)
74  {
75      ui->pushButton_ransac->setEnabled(status);
76      ui->pushButton_lmeds->setEnabled(status);
77  }
78
79
80  void DialogCompare::DisplayCompare(std::vector< DMatch > matches, std::
        vector<KeyPoint> one, std::vector<KeyPoint> two ){
81      //-- Draw only "good" matches
82      Mat img_matches;
83      if(matches.size()>0){
84          corros_img1.clear();
85          corros_img2.clear();
86          for( size_t i = 0; i < matches.size(); i++ )
87          {
88            //-- Get the keypoints from the good matches
89            corros_img1.push_back( one[ matches[i].queryIdx ].pt );
90            corros_img2.push_back( two[ matches[i].trainIdx ].pt );
```

```
91          }
92      }
93      if(corros_img1.size()>3){
94          ToggleAlignment(true);
95      }
96      else{
97          ToggleAlignment(false);
98      }
99      drawMatches( images_src_[0], one, images_src_[1], two,
100                 matches, img_matches, Scalar::all(-1), Scalar::all(-1),
101                 std::vector<char>(), DrawMatchesFlags::
                       NOT_DRAW_SINGLE_POINTS );
102
103     //-- Show detected matches
104
105     ui->label->setPixmap(QPixmap::fromImage(MatToQImage(img_matches)).
106                                 scaled(ui->label->width(),
107                                        ui->label->height(),Qt::
                                            KeepAspectRatio));
108 }
109
110 DialogCompare::~DialogCompare()
111 {
112     delete ui;
113 }
114
115 void DialogCompare::on_pushButton_clicked()
116 {
117     this->close();
118 }
119 void DialogCompare::MatchFLANN(std::vector<KeyPoint> keypoints_1,
120                                std::vector<KeyPoint> keypoints_2,
121                                Mat descriptors_1, Mat descriptors_2)
                                    {
122
123     FlannBasedMatcher matcher;
```

```
124     std::vector< DMatch > matches;
125     matcher.match( descriptors_1, descriptors_2, matches );
126     double min_dist = 100;
127     for( int i = 0; i < descriptors_1.rows; i++ )
128     { double dist = matches[i].distance;
129       if( dist < min_dist ) min_dist = dist;
130     }
131     std::vector< DMatch > good_matches;
132     for( int i = 0; i < descriptors_1.rows; i++ )
133     { if( matches[i].distance <= max(2*min_dist, 0.02) )
134       { good_matches.push_back( matches[i]); }
135     }
136     DisplayCompare(good_matches, keypoints_1, keypoints_2);
137 }
138 void DialogCompare::MatchBFM(std::vector<KeyPoint> keypoints_1,
139                              std::vector<KeyPoint> keypoints_2,
140                              Mat descriptors_1, Mat descriptors_2){
141     BFMatcher matcher(NORM_L1);
142     std::vector< DMatch > matches;
143     matcher.match( descriptors_1, descriptors_2, matches );
144     double  min_dist = 100;
145     for( int i = 0; i < descriptors_1.rows; i++ )
146     { double dist = matches[i].distance;
147       if( dist < min_dist ) min_dist = dist;
148     }
149     std::vector< DMatch > good_matches;
150     for( int i = 0; i < descriptors_1.rows; i++ )
151     { if( matches[i].distance <= max(2*min_dist, 0.02) )
152       { good_matches.push_back( matches[i]); }
153     }
154     DisplayCompare(good_matches, keypoints_1, keypoints_2);
155 }
156
157
158
159 inline float Point2d_d(Point2f a, Point2f b){
```

```
160     return sqrt(pow((b.x-a.x),2)+pow((b.y-a.y),2));
161 }
162 inline double Point2d_d_s(P2d a, P2d b){
163     return sqrt(pow((b.x-a.x),2)+pow((b.y-a.y),2));
164     return fabs(a.x - b.x)+fabs(a.y - b.y);
165 }
166 inline double Point2d_d_s(Point2f a, Point2f b){
167     return sqrt(pow((b.x-a.x),2)+pow((b.y-a.y),2));
168     return fabs(a.x - b.x)+fabs(a.y - b.y);
169 }
170 std::vector<TriangleG> DialogCompare::FindAllTriangles_GROTH(std::vector
    <Point2f> mc2, int x, int y){
171
172     std::vector<Point2f> mc1;
173     Point2f pof;
174     if(x == 0 || y == 0)
175         mc1 = mc2;
176     else
177         for( uint i = 0; i< mc2.size(); ++i ){
178             pof.x = mc2[i].x/x;
179             pof.y = mc2[i].y/y;
180             mc1.push_back(pof);
181         }
182     int blob_count = mc2.size();
183     blob_count = (blob_count*(blob_count-1)*(blob_count-2))/6;
184     std::vector<TriangleG> triangle_collection;
185     TriangleG triangle;
186     double side1, side2, side3;
187     Point2d v1, v2, v3;
188     double tol, E = 0.0005;
189     double R, S, r3, r2, t_pow_2_r, C, t_pow_2_c, x3, x2, y3, y2;
190     for( uint i = 0; i< mc1.size(); ++i ){
191         for( uint j = i+1; j< mc1.size(); ++j ){
192             for(uint h = j+1; h < mc1.size(); ++h){
193
194                 side1 = Point2d_d(mc1[i], mc1[j]);
```

```
195                 side2 = Point2d_d(mc1[j], mc1[h]);
196                 side3 = Point2d_d(mc1[h], mc1[i]);
197
198                 if(side1 > side2 && side2 > side3){
199                     v1 = mc1[i]; v2 = mc1[h]; v3 = mc1[j];
200                     triangle.v1 = mc2[i];     triangle.v2 = mc2[h];
                            triangle.v3 = mc2[j];
201                 }else
202                 if(side1 > side3 && side3 > side2){
203                     v1 = mc1[j]; v2 = mc1[h]; v3 = mc1[i];
204                     triangle.v1 = mc2[j];     triangle.v2 = mc2[h];
                            triangle.v3 = mc2[i];
205                 }else
206                 if(side2 > side1 && side1 > side3){
207                     v1 = mc1[h]; v2 = mc1[i]; v3 = mc1[j];
208                     triangle.v1 = mc2[h];     triangle.v2 = mc2[i];
                            triangle.v3 = mc2[j];
209                 }else
210                 if(side2 > side3 && side3 > side1){
211                     v1 = mc1[j]; v2 = mc1[i]; v3 = mc1[h];
212                     triangle.v1 = mc2[j];     triangle.v2 = mc2[i];
                            triangle.v3 = mc2[h];
213                 }else
214                 if(side3 > side1 && side1 > side2){
215                     v1 = mc1[h]; v2 = mc1[j]; v3 = mc1[i];
216                     triangle.v1 = mc2[h];     triangle.v2 = mc2[j];
                            triangle.v3 = mc2[i];
217                 }else{//if(dis3 > dis2 && dis2 > dis1){
218                     v1 = mc1[i]; v2 = mc1[j]; v3 = mc1[h];
219                     triangle.v1 = mc2[i];     triangle.v2 = mc2[j];
                            triangle.v3 = mc2[h];
220
221                 }
222
223                x3 = v3.x - v1.x;
224                x2 = v2.x - v1.x;
```

```
225
226                 y3 = v3.y - v1.y;
227                 y2 = v2.y - v1.y;
228
229                 r3 = sqrt((x3*x3)+(y3*y3));
230                 r2 = sqrt((x2*x2)+(y2*y2));
231
232                 R  = r3/r2;
233                 if(R < 20){
234                     C = ((x3*x2) + (y3*y2))/(r3*r2);
235                     S = (1 - (C*C)); //sin x + cos x = 1
236
237                     tol= E*E*((1/(r3*r3)) - (C/(r3*r2)) + (1/(r2*r2)));
238
239                     t_pow_2_r = 2*R*R*tol;
240
241                     t_pow_2_c = (2*S*S*tol) + (3*C*C*tol*tol);
242                     triangle.C = C;
243                     triangle.R = R;
244                     triangle.t2C = t_pow_2_c;
245                     triangle.t2R = t_pow_2_r;
246                     triangle.v1n = v1;    triangle.v2n = v2; triangle.v3n
                        = v3;
247                     triangle_collection.push_back(triangle);
248
249                 }
250             }
251
252
253         }
254
255     }
256
257     return triangle_collection;
258 }
259
```

```
260 void DialogCompare::MatchLoopTriangles_GROTH(std::vector<TriangleG> a,
        std::vector<TriangleG> b){
261     a_f_g.swap(a);
262     b_f_g.swap(b);
263
264     #pragma omp parallel for
265     for(size_t i = 0; i < a_f_g.size(); ++i){
266         for(size_t j = 0; j < b_f_g.size(); ++j){
267             if(a_f_g[i]==b_f_g[j]){
268
269             }
270         }
271     }
272     std::sort(a_f_g.begin(), a_f_g.end(), ComparisonDistance());
273
274 }
275
276 std::vector<TriangleV> DialogCompare::FindAllTriangles_SAFAR(std::vector
        <Point2f> mc1, bool fit){
277     TriangleV store_t;
278     double side1, side2, side3;
279     double a=0, b=0, c=0;
280     Point2f v1, v2, v3;
281     std::vector<TriangleV> triangle_collection;
282
283     for( uint i = 0; i< mc1.size(); ++i ){
284         for( uint j = i+1; j< mc1.size(); ++j ){
285             for(uint h = j+1; h < mc1.size(); ++h){
286                 side1 = Point2d_d(mc1[i], mc1[j]);
287                 side2 = Point2d_d(mc1[j], mc1[h]);
288                 side3 = Point2d_d(mc1[h], mc1[i]);
289
290                 //Sorting the sides of the triangle
291                 if(side1 > side2 && side2 > side3){
292                     a = side1; b = side2; c = side3;
293                     v1 = mc1[i]; v2 = mc1[h]; v3 = mc1[j];
```

```
294                 }else
295                 if(side1 > side3 && side3 > side2){
296                    a = side1; b = side3; c = side2;
297                    v1 = mc1[j]; v2 = mc1[h]; v3 = mc1[i];
298                 }else
299                 if(side2 > side1 && side1 > side3){
300                    a = side2; b = side1; c = side3;
301                    v1 = mc1[h]; v2 = mc1[i]; v3 = mc1[j];
302                 }else
303                 if(side2 > side3 && side3 > side1){
304                    a = side2; b = side3; c = side1;
305                    v1 = mc1[j]; v2 = mc1[i]; v3 = mc1[h];
306                 }else
307                 if(side3 > side1 && side1 > side2){
308                    a = side3; b = side1; c = side2;
309                    v1 = mc1[h]; v2 = mc1[j]; v3 = mc1[i];
310                 }else{//if(dis3 > dis2 && dis2 > dis1){
311                    a = side3; b = side2; c = side1;
312                    v1 = mc1[i]; v2 = mc1[j]; v3 = mc1[h];
313                 }
314                 // if neighbors number is less than the cell count
315                 if(fit){
316                     std::vector<double> list;
317                     // mesure the distance between current triangle and
                            all cells
318                     for(uint n = 0; n < mc1.size(); ++n){
319                         if(n!=i && n!=j && n!=h)
320                         list.push_back(Point2d_d_s(v1, mc1[n]));
321                     }
322                     // sort the measured distances
323                     std::sort(list.begin(), list.end());
324
325                     store_t.list.resize(N_SIZE); // bypass a bug in STL
326
327                     // store only the closest N neighbors
328                     for(uint n=0; n<N_SIZE&&n<list.size(); ++n){
```

```
329                         store_t.list[n] = list[n]/a;
330                     }
331                 }
332                 // storing the center of the triangle
333                 store_t.pos.x = b/a;
334                 store_t.pos.y = c/a;
335                 // storing the triangle vertices
336                 store_t.v1 = v1;
337                 store_t.v2 = v2;
338                 store_t.v3 = v3;
339                 triangle_collection.push_back(store_t);
340             }
341
342
343         }
344
345     }
346     return triangle_collection;
347 }
348
349 std::vector<TriangleV> DialogCompare::FindAllTriangles_VALDES(std::
    vector<Point2f> mc1){
350     TriangleV store_t;
351     double side1, side2, side3;
352     double a=0, b=0, c=0;
353     Point2f v1, v2, v3;
354     std::vector<TriangleV> triangle_collection;
355
356     for( uint i = 0; i< mc1.size(); ++i ){
357         for( uint j = i+1; j< mc1.size(); ++j ){
358             for(uint h = j+1; h < mc1.size(); ++h){
359                 side1 = Point2d_d(mc1[i], mc1[j]);
360                 side2 = Point2d_d(mc1[j], mc1[h]);
361                 side3 = Point2d_d(mc1[h], mc1[i]);
362
363                 if(side1 > side2 && side2 > side3){
```

```
364                     a = side1; b = side2; c = side3;
365                     v1 = mc1[i]; v2 = mc1[h]; v3 = mc1[j];
366                 }else
367                 if(side1 > side3 && side3 > side2){
368                     a = side1; b = side3; c = side2;
369                     v1 = mc1[j]; v2 = mc1[h]; v3 = mc1[i];
370                 }else
371                 if(side2 > side1 && side1 > side3){
372                     a = side2; b = side1; c = side3;
373                     v1 = mc1[h]; v2 = mc1[i]; v3 = mc1[j];
374                 }else
375                 if(side2 > side3 && side3 > side1){
376                     a = side2; b = side3; c = side1;
377                     v1 = mc1[j]; v2 = mc1[i]; v3 = mc1[h];
378                 }else
379                 if(side3 > side1 && side1 > side2){
380                     a = side3; b = side1; c = side2;
381                     v1 = mc1[h]; v2 = mc1[j]; v3 = mc1[i];
382                 }else{//if(dis3 > dis2 && dis2 > dis1){
383                     a = side3; b = side2; c = side1;
384                     v1 = mc1[i]; v2 = mc1[j]; v3 = mc1[h];
385                 }
386                 store_t.pos.x = b/a;
387                 store_t.pos.y = c/a;
388                 store_t.v1 = v1;
389                 store_t.v2 = v2;
390                 store_t.v3 = v3;
391                 triangle_collection.push_back(store_t);
392             }
393 
394 
395         }
396 
397     }
398     return triangle_collection;
399 }
```

```
400
401
402 void DialogCompare::MatchLoopTriangles_SAFAR(std::vector<TriangleV> a,
        std::vector<TriangleV> b, bool fit){
403     a_f_v.swap(a); // smaller
404     b_f_v.swap(b);
405
406     // if neighbors number is less than the cell count
407     if(fit){
408         #pragma omp parallel for
409         for(size_t i = 0; i < a_f_v.size(); ++i){
410             for(size_t j = 0; j < b_f_v.size(); ++j){
411                 // for every combination of triangles from both sets
                        measure the error rate
412                 // using the top neighbors for each triangle
413                 double n_error_rate = 0;
414                 if(!a_f_v.empty() && !b_f_v.empty())
415                     for(uint n=0; n<N_SIZE; ++n){
416                         n_error_rate += abs(a_f_v[i].list[n] - b_f_v[j].
                                list[n]);
417                     }
418                 double distance = Point2d_d_s(a_f_v[i].pos, b_f_v[j].pos
                        ) + n_error_rate/10;
419                 // if a better match found store it
420                 if(distance < a_f_v[i].distance){
421                     a_f_v[i].distance = distance;
422                     a_f_v[i].match = &b_f_v[j];
423                 }
424
425             }
426         }
427     }else{
428         #pragma omp parallel for
429         for(size_t i = 0; i < a_f_v.size(); ++i){
430             for(size_t j = 0; j < b_f_v.size(); ++j){
```

```
431                     double distance = Point2d_d_s(a_f_v[i].pos, b_f_v[j].pos
                            );
432                     if(distance < a_f_v[i].distance){
433                         a_f_v[i].distance = distance;
434                         a_f_v[i].match = &b_f_v[j];
435                     }
436 
437                 }
438             }
439         }
440 
441         std::sort(a_f_v.begin(), a_f_v.end(), ComparisonDistance());
442 }
443 
444 
445 void DialogCompare::MatchLoopTriangles_VALDES(std::vector<TriangleV> a,
        std::vector<TriangleV> b){
446         a_f_v.swap(a); // smaller
447         b_f_v.swap(b);
448 
449         #pragma omp parallel for
450         for(size_t i = 0; i < a_f_v.size(); ++i){
451             for(size_t j = 0; j < b_f_v.size(); ++j){
452                 double distance = Point2d_d_s(a_f_v[i].pos, b_f_v[j].pos);
453                 if(distance < a_f_v[i].distance){
454                     a_f_v[i].distance = distance;
455                     a_f_v[i].match = &b_f_v[j];
456                 }
457 
458             }
459         }
460         std::sort(a_f_v.begin(), a_f_v.end(), ComparisonDistance());
461 }
462 template <class G_TRI>
463 void DialogCompare::RateNSliderS(std::vector<G_TRI>& list_of_tri, Mat *
        img_1, Mat *img_2, int top, bool reverse)
```

```
464 {
465     RNG rng(12345);
466     int start, array_size(list_of_tri.size());
467     if(top > array_size){
468         top = array_size;
469     }
470     if(reverse){
471         start = array_size- top;
472         top   = array_size; // flip for reverse
473     }else{
474         start = 0;
475     }
476     corros_img1.clear();
477     corros_img2.clear();
478
479     for(int i = start; i < top; ++i){
480         if(list_of_tri[i].match == NULL) continue;
481         Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255),
                rng.uniform(0,255) );
482         //debugging (drawing circles to see matches)
483         if(debug_mode){
484             Scalar color0 = Scalar( rng.uniform(0, 255), rng.uniform
                    (0,255), rng.uniform(0,255) );
485             Scalar color1 = Scalar( rng.uniform(0, 255), rng.uniform
                    (0,255), rng.uniform(0,255) );
486             Scalar color2 = Scalar( rng.uniform(0, 255), rng.uniform
                    (0,255), rng.uniform(0,255) );
487
488             int radius = 10;
489             circle(*img_1, list_of_tri[i].v1,radius, color0,CV_FILLED,
                    8, 0);
490             circle(*img_1, list_of_tri[i].v2,radius, color1,CV_FILLED,
                    8,0);
491             circle(*img_1, list_of_tri[i].v3,radius, color2,CV_FILLED,
                    8,0);
```

```
492             circle(*img_2, list_of_tri[i].match->v1,radius, color0,
                    CV_FILLED, 8,0);
493             circle(*img_2, list_of_tri[i].match->v2,radius, color1,
                    CV_FILLED, 8,0);
494             circle(*img_2, list_of_tri[i].match->v3,radius, color2,
                    CV_FILLED, 8,0);
495         }
496
497         corros_img1.push_back(list_of_tri[i].v1);
498         corros_img2.push_back(list_of_tri[i].match->v1);
499
500         corros_img1.push_back(list_of_tri[i].v2);
501         corros_img2.push_back(list_of_tri[i].match->v2);
502
503         corros_img1.push_back(list_of_tri[i].v3);
504         corros_img2.push_back(list_of_tri[i].match->v3);
505         Point2d centerA, centerMA;
506         centerA.x = (list_of_tri[i].v1.x + list_of_tri[i].v2.x +
                list_of_tri[i].v3.x) / 3;
507         centerA.y = (list_of_tri[i].v1.y + list_of_tri[i].v2.y +
                list_of_tri[i].v3.y) / 3;
508         centerMA.x = (list_of_tri[i].match->v1.x + list_of_tri[i].match
                ->v2.x + list_of_tri[i].match->v3.x) / 3;
509         centerMA.y = (list_of_tri[i].match->v1.y + list_of_tri[i].match
                ->v2.y + list_of_tri[i].match->v3.y) / 3;
510
511
512         centerA.x = (list_of_tri[i].v1.x + list_of_tri[i].v2.x +
                list_of_tri[i].v3.x) / 3;
513         centerA.y = (list_of_tri[i].v1.y + list_of_tri[i].v2.y +
                list_of_tri[i].v3.y) / 3;
514         centerMA.x = (list_of_tri[i].match->v1.x + list_of_tri[i].match
                ->v2.x + list_of_tri[i].match->v3.x) / 3;
515         centerMA.y = (list_of_tri[i].match->v1.y + list_of_tri[i].match
                ->v2.y + list_of_tri[i].match->v3.y) / 3;
516         corros_img1.push_back(centerA);
```

```
517             corros_img2.push_back(centerMA);
518             line(*img_1, list_of_tri[i].v1, list_of_tri[i].v3, color, 3,
                    CV_AA);
519             line(*img_1, list_of_tri[i].v1, list_of_tri[i].v2, color, 3,
                    CV_AA);
520             line(*img_1, list_of_tri[i].v2, list_of_tri[i].v3, color, 3,
                    CV_AA);
521
522             line(*img_2, list_of_tri[i].match->v1, list_of_tri[i].match->v3,
                     color, 3, CV_AA);
523             line(*img_2, list_of_tri[i].match->v1, list_of_tri[i].match->v2,
                     color, 3, CV_AA);
524             line(*img_2, list_of_tri[i].match->v2, list_of_tri[i].match->v3,
                     color, 3, CV_AA);
525
526         }
527 }
528 void DialogCompare::moving_slider(){
529     Mat img_2;
530     Mat img_1;
531     if(flip){
532         img_2 = images_src_[1].clone();
533         img_1 = images_src_[0].clone();
534     }else{
535          img_2 = images_src_[0].clone();
536          img_1 = images_src_[1].clone();
537     }
538     if(algorithm == AlgorithmsName::alg_GROTH)
539          RateNSliderS(a_f_g, &img_2, &img_1, ui->lcdNumber_rate->value(),
                     ui->checkBox->isChecked());
540     else if(algorithm == AlgorithmsName::alg_SAFAR || algorithm ==
            AlgorithmsName::alg_VALDES)
541          RateNSliderS(a_f_v, &img_2, &img_1, ui->lcdNumber_rate->value(),
                     ui->checkBox->isChecked());
542     DisplayCompare(std::vector< DMatch >(), std::vector<KeyPoint>(), std
            ::vector<KeyPoint>());
```

```
543     Mat img_matches;
544     drawMatches( img_1, std::vector<KeyPoint>(), img_2, std::vector<
            KeyPoint>(),
545                  std::vector< DMatch >(), img_matches, Scalar::all(-1),
                         Scalar::all(-1),
546                  std::vector<char>(), DrawMatchesFlags::
                         NOT_DRAW_SINGLE_POINTS );
547
548     //-- Show detected matches
549
550     ui->label->setPixmap(QPixmap::fromImage(MatToQImage(img_matches)).
551                                         scaled(ui->label->width(),
552                                                ui->label->height(),Qt::
                                                    KeepAspectRatio));
553
554 }
555
556 void DialogCompare::MatchGROTH(std::vector<Point2f> mc1, std::vector<
        Point2f> mc2)
557 {
558     std::vector<TriangleG> t_mc1(FindAllTriangles_GROTH(mc1, images_src_
            [0].cols, images_src_[0].rows));
559     std::vector<TriangleG> t_mc2(FindAllTriangles_GROTH(mc2, images_src_
            [1].cols, images_src_[1].rows));
560
561     if(t_mc1.size() < t_mc2.size()){
562         MatchLoopTriangles_GROTH(t_mc1, t_mc2);
563         flip = false;
564     }
565     else{
566         MatchLoopTriangles_GROTH(t_mc2, t_mc1);
567         flip = true;
568     }
569     moving_slider();
570 }
571
```

```
572
573 void DialogCompare::MatchSAFAR(std::vector<Point2f> mc1, std::vector<
    Point2f> mc2)
574 {
575     try {
576         // check if the provided neighbors number is less than the cell
              count
577         // if it is less it will ignore KNN part of the algorithm which
              will become more like Valdes
578         bool fit = ((mc1.size()-3)>N_SIZE)&&((mc2.size()-3)>N_SIZE) ?
              true : false;
579
580         //Finding all possiable triangles and storing them
581         std::vector<TriangleV> t_mc1(FindAllTriangles_SAFAR(mc1, fit));
582         std::vector<TriangleV> t_mc2(FindAllTriangles_SAFAR(mc2, fit));
583         if(t_mc1.size() < t_mc2.size()){
584             MatchLoopTriangles_SAFAR(t_mc1, t_mc2, fit);
585             flip = false;
586         }
587         else{
588             MatchLoopTriangles_SAFAR(t_mc2, t_mc1, fit);
589             flip = true;
590         }
591
592
593
594
595         moving_slider();
596     }
597     catch (const std::bad_alloc&) {
598       std::cout<<"Not enough memory"<<std::endl;
599     }
600 }
601
602 void DialogCompare::MatchVALDES(std::vector<Point2f> mc1, std::vector<
    Point2f> mc2)
```

```
603 {
604     try {
605         std::vector<TriangleV> t_mc1(FindAllTriangles_VALDES(mc1));
606         std::vector<TriangleV> t_mc2(FindAllTriangles_VALDES(mc2));
607 
608         if(t_mc1.size() < t_mc2.size()){
609             MatchLoopTriangles_VALDES(t_mc1, t_mc2);
610             flip = false;
611         }
612         else{
613             MatchLoopTriangles_VALDES(t_mc2, t_mc1);
614             flip = true;
615         }
616 
617 
618         moving_slider();
619     }
620     catch (const std::bad_alloc&) {
621       std::cout<<"Not enough memory"<<std::endl;
622     }
623 }
624 
625 void DialogCompare::Alignment(int method)
626 {
627     // if more than 3 corresponding points where found
628     if(corros_img1.size() > 3){
629         // create the transformation matrix
630         Mat H = findHomography(corros_img1, corros_img2, method);
631         // if the matrix is valid
632         if(H.rows>0 && H.cols>0){
633             Mat src,src2;
634             if(!flip){
635                 src = images_src_[0].clone();
636                 src2 = images_src_[1].clone();
637             }else{
638                  src = images_src_[1].clone();
```

```
639             src2 = images_src_[0].clone();
640         }
641         std::vector<Point2f> obj_corners(4);
642         obj_corners[0] = Point(0,0);
643         obj_corners[1] = Point( src.cols, 0 );
644         obj_corners[2] = Point( src.cols, src.rows );
645         obj_corners[3] = Point( 0, src.rows );
646         std::vector<Point2f> scene_corners(4);
647         perspectiveTransform( obj_corners, scene_corners, H);
648         Mat subed, img_matched = src.clone();
649         warpPerspective(src, img_matched, H, src.size());
650         cv::add(img_matched,src2,subed);
651
652
653         ///
654         ///          Data set (extracting the pixels count)
655         ///
656
657         cv::Mat img_matched_hsv_green, img_matched_hsv_red, src2_hsv
                , subed_hsv;
658         cv::Scalar Green(0, 255, 0), Red(0, 0, 255);
659         cvtColor(subed, subed_hsv, CV_BGR2HSV);
660
661         float yellow_count, yellow_pixels, red_count_after,
                red_pixels, red_pixels_after, green_count_after,
                green_pixels, green_pixels_after;
662         std::vector<std::vector<Point> > contours;
663         std::vector<Vec4i> hierarchy;
664         if(flip){
665             inRange(img_matched, Red, Red, img_matched_hsv_red); //
                    red
666             red_pixels      = cv::countNonZero(img_matched_hsv_red
                    >128);
667             findContours( img_matched_hsv_red, contours, hierarchy,
                    CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0,
                    0) );
```

```
668                 red_count_after = contours.size();
669                 inRange(src2, Green, Green, src2_hsv);
670                 green_pixels   = cv::countNonZero(src2_hsv > 128);
671                 findContours( src2_hsv, contours, hierarchy,
                        CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0,
                        0) );
672                 green_count_after = contours.size();
673             }else{
674                 inRange(img_matched, Green, Green, img_matched_hsv_green
                        ); // green
675                 green_pixels      = cv::countNonZero(
                        img_matched_hsv_green>128);
676                 findContours( img_matched_hsv_green, contours, hierarchy
                        , CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0,
                         0) );
677                 green_count_after = contours.size();
678                 inRange(src2, Red, Red, src2_hsv);
679                 red_pixels   = cv::countNonZero(src2_hsv > 128);
680                 findContours( src2_hsv, contours, hierarchy,
                        CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0,
                        0) );
681                 red_count_after = contours.size();
682             }
683
684             inRange(subed, Red, Red, img_matched_hsv_red); // red
685             inRange(subed, Green, Green, img_matched_hsv_green); //
                    green
686             inRange(subed_hsv, cv::Scalar(20, 100, 100), cv::Scalar(30,
                    255, 255), subed_hsv); // yellow
687             yellow_pixels      = cv::countNonZero(subed_hsv);
688             red_pixels_after   = cv::countNonZero(img_matched_hsv_red);
689             green_pixels_after = cv::countNonZero(img_matched_hsv_green)
                    ;
690
691
```

```
692             findContours( subed_hsv, contours, hierarchy,
                    CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
693             yellow_count = contours.size();
694
695
696             store_them =     QString::number(red_count)+","+
697                              QString::number(red_pixels)+","+
698                              QString::number(red_count_after)+","+
699                              QString::number(red_pixels_after)+","+
700                              QString::number(green_count)+","+
701                              QString::number(green_pixels)+","+
702                              QString::number(green_count_after)+","+
703                              QString::number(green_pixels_after)+","+
704                              QString::number(yellow_pixels)+","+
705                              QString::number(yellow_count);
706             AdaBoost classifier;
707             classifier.Evaluate(yellow_count/green_count_after,
708                                 yellow_pixels/green_pixels,
709                                 yellow_pixels/((red_pixels+green_pixels)
                                        /2.0),
710                                 yellow_pixels/((red_pixels_after+
                                        green_pixels_after)/2.0),
711                                 yellow_pixels/red_pixels_after);
712
713
714             putText(subed, classifier.isAligned()?"Aligned":"Not Aligned
                    ", Point(10,30), cv::FONT_HERSHEY_SIMPLEX, 1, Scalar
                    (255), 2, LINE_AA);
715
716
717             ui->textEdit_aligend->setText(classifier.print().c_str());
718             ui->label->setPixmap(QPixmap::fromImage(MatToQImage(subed)).
719                                             scaled(ui->label->width(),
720                                                     ui->label->height(),Qt::
                                                        KeepAspectRatio));
721         }
```

```
722         }
723 }
724
725 void DialogCompare::on_pushButton_ransac_clicked()
726 {
727         Alignment(CV_RANSAC);
728 }
729
730 void DialogCompare::on_pushButton_lmeds_clicked()
731 {
732         Alignment(CV_LMEDS);
733 }
734
735 void DialogCompare::on_spinBox_true_valueChanged(int arg1)
736 {
737         ui->spinBox_false->setValue(abs(arg1-10));
738 }
739
740 void DialogCompare::on_spinBox_false_valueChanged(int arg1)
741 {
742         ui->spinBox_true->setValue(abs(arg1-10));
743 }
```

Listing C.3: DialogCompare Class Code