Technical University of Crete
School of Electrical and Computer Engineering

THESIS

SPECTRAL IMAGING AND MACHINE LEARNING TECHNOLOGIES FOR AUTOMATION OF DIAGNOSTIC MICROSCOPY

Gialitakis Emmanouil

Supervisor:	Professor Costas Balas
Thesis committee:	Professor Costas Balas
	Professor Michalis Zervakis
	Associate Professor Michail G. Lagoudakis

Chania, August 2021

Abstract

Microscopy for years is an instrumental technology for analyzing tissue samples and locating cancerous cells. The biopsy is a process that can last for several days and is crucial, since the doctor will decide on the most suitable treatment depending on the results. The goal of this thesis is to speed up the process of analyzing a biopsy by using image stitching algorithms. By creating high-resolution mosaics of the samples, it will be easier for different doctors to examine the same sample, while being located in a different area or re-examine the same sample, if needed. The chosen algorithm for stitching is SIFT, which is distinguishable among other algorithms due to its high accuracy. At the same time, the main disadvantage that needs to be mended is the time needed to complete a stitching due to its high complexity. Using various techniques aiming to reduce the elapsed time, like reducing the image size to be analyzed each time and defining regions of interest in the images, can reduce the time needed. By applying those techniques, it is possible to speed up the average time of a single horizontal stitch by approximately four times. These results suggest that it will be viable for the algorithm to be used in a microscope, reducing the time of analysis of the samples.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BRIEF	Binary Robust Independent Elementary Features
CPU	Central processing unit
DoG	Difference of Gaussians
DoL	Difference of Laplacians
FAST	Features from Accelerated Segment Test
FS	Full Search
GPU	Graphics processing unit
MSER	Maximally Stable Extremal Regions
OpenCV	Open-Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
RANSAC	Random sample consensus
ROI	Regions of interest
SAD	Sum of absolute differences
SIFT	Scale Invariant Feature Transform
SSD	Sum of the squared differences
SURF	Speeded-Up Robust Features

Acknowledgements

First of all, I would like to express my deepest appreciation to my supervisor, Prof. Costas Ballas, for his continuous and invaluable support and guidance throughout the work of this thesis.

Furthermore, I would like to thank all the members of the Electronics Lab for the immediate help provided whenever needed. Moreover, I am deeply thankful to my friends and colleagues in the lab that helped me come up with new ideas and insights on solving numerous problems that would come up through the work of the thesis.

Finally, I would like to express my deepest gratitude to my family and friends who have always been supportive throughout the years of my studies.
My sincerest thanks to all of you.

1 Introduction

This chapter outlines the background of microscopy and image stitching, as well as introduces the problem that we aim to solve in section 1. It also includes a brief mention of the goals of the thesis (section 2) and its purposes. Finally, section 1.3 includes an outline of the remaining chapters of the thesis.

1.1 MICROSCOPY & IMAGE STITCHING

A microscope is a scientific tool used to examine any object that is too small to be seen by the naked eye. It is a tool of paramount importance that has aided the development of multiple scientific fields for many decades, from examining forensic evidence, whereby observing striations in bullets can help determine which gun was used to aid the study of the surface of individual atoms. However, perhaps the scientific field influenced most from its invention is the medical field, where the use of the microscope helped understand the structures of the cells and even the functions of the proteins within the cells. Furthermore, it has an important role when a doctor needs to examine tissue, which comes from a biopsy, in order to decide whether the sample contains cancerous cells. After the doctor obtains the tissue sample, it is sent to a lab for analysis. Usually, the sample goes through some processing before being analyzed, which includes freezing to preserve the sample, it may get chemically treated and is sliced into very thin sections. Those sections of the sample are placed on glass slides and sometimes are stained with certain chemical substances to enhance some features of the cell that can provide important information, like the cell membrane and the nucleus. At this point, the sample is inserted into the microscope in order to be examined. By observing the results, the doctor can determine where the cancer was originated as well as categorize it depending on how aggressive it is, which is a crucial step towards deciding the type of treatment that will follow. Since cancer has such a wide variety of types (places to originate), it is not always possible for a single doctor to examine the tissue and produce the results. Doctors from different fields, such as a hematologist or some other specialty, need to consult and review the sample in order to produce the results. Even in cases where a single doctor can perform the examination alone, there may be a need for a re-examination by the doctor or colleague to reassure the results. In the end, the whole process takes a long time to complete, and

even though on some rare occasions, the results can be produced within minutes or hours from the time the sample was collected, generally several days will be needed to produce results. At this point, the current thesis aims to assist the doctors, not in a way to change the standard procedure of the examination, but hopefully reduce the time needed between the acquisition of the tissue and the final results.

Image stitching, otherwise known as mosaicing, is the process where two or more images that have some degree of overlap are merged into a segmented panorama or high-resolution image. Algorithms for image aligning and stitching are widely used in computer vision for several years. Some date back to 1981 when Lucas and Kanade [1] introduced a technique known as optical flow. Such algorithms are widely used today, from image video stabilization, where aligning is applied for each frame, to creation of digital maps from satellite images using image stitching techniques. With the increasingly widespread use of smartphones, those algorithms are necessary since every camera needs a non-mechanical stabilization solution, as well as the panoramic photography mode.

1.2 CONTEXT

The purpose of this thesis is to create an image stitching application that will be designed for a fully automated microscope. The achievement of this goal will be based on the usage of the SIFT algorithm, which was first introduced by D.Lowe in 2004 [2]. Although this algorithm is not a recent development, it has features that distinguish it, even among algorithms that were more recently created. An important characteristic of this method is that it is a feature-based algorithm that has certain advantages in contrast to other algorithms, the details of which are going to be analyzed in a following chapter. The utilization of a stitching algorithm for images taken from the microscope is not a new invention. This thesis will examine the results of using the SIFT algorithm, whose main disadvantage over similar features-based methods is its computational cost. By applying certain techniques, hopefully, the results will be suitable to be utilized in a microscope without delaying the whole procedure.

1.3 THESIS OUTLINE

The present thesis is divided into six sections. The first chapter is an introduction to the problem that the thesis aims to solve, along with some background information about microscopy and image stitching. In chapter 2, we delve deeper into the algorithms that were considered to be used, as well as point out the reasons that helped decide on the algorithms to be implemented. Chapter 3 further analyses the algorithms that were chosen and also analysis their implementation. Additionally, it contains brief information about the tools that were used in the thesis (e.g., the microscope). In chapter 4, the foci are to analyze the results of the tests that were performed on the final application. Lastly, chapter 5 includes the conclusions that were drawn during the development of the thesis, as well as some proposals for future work.

2 Literature Review

In this chapter, we will analyze the algorithms that were used in the thesis. In the first section, there is a brief summary of algorithms that exist and the reason behind the selection of the algorithm that was finally used. Consequently, the next section will delve deeper into the chosen algorithm and analyze it from a theoretical perspective. Section 3 will analyze the matching process and examine the theoretical background of the algorithms to be used. Finally, section 4 includes the algorithms of the blending process.

2.1 IMAGE STITCHING ALGORITHMS

First of all, it is important to specify which algorithm was chosen for this task and what characteristics distinguished him among others, designed for solving the same problem (image stitching). The final choice was SIFT, which was developed by D.Lowe [2], and the primary advantage that this particular algorithm possesses is the high accuracy. According to research published by Bonny and Uddin [3], who compared a variety of feature-based methods, including SURF (Speeded Up Robust Features), FAST (Features from Accelerated Segment Test), Harris corner detector, and MSER (Maximally Stable Extremal Regions), they concluded that among the algorithms enlisted in their paper, SURF seems to have the highest accuracy. However, they did not include the SIFT algorithm in their research. But according to Karami, Prasad, and Shehata [4], who dived deeper into the comparison of SIFT, SURF, and ORB (oriented FAST rotated BRIEF), have concluded that although both SURF and ORB are generally faster, they cannot achieve the high match rate if SIFT. Especially ORB can have up to 19,2% deviation from the high scoring SIFT when it comes to the match rate, and the only occasion where ORB accomplishes both a higher match rate and speed is when comparing images with different scale, a scenario that generally is not expected to be seen in the current application. Similarly, the SURF algorithm, although being faster in every experiment, the only instance where its score is higher than that of SIFT, is when the images have a different scale. It should also be noted that the experiment concerning the scale of the images was implemented, having one of them being scaled two times.

2.2 SIFT ANALYSIS

This particular algorithm was originally designed in order to extract features from images so that it may be capable of reliably matching objects that are located inside, according to the data of a database. During its creation, techniques were applied so that the algorithm will have tolerance to noise contained in the images, to some differences in luminance, as well as a certain indulgence to affine distortions. It is important to note that the whole procedure requires the use of grayscale images.

2.2.1 Keypoint Detection

The first step towards the specification of the image's features is the detection of the scale-space extrema, i.e., the pixels with the maximum value in comparison to the surrounding pixels, which refers to the scale portion, along with the extrema between scales. In order to accomplish that, Gaussian filters were applied repeatedly with an increasing standard deviation of the gaussian distribution, in addition to a series of rescales of the initial image. The result of the described process is the creation of a series of octaves, which consist of a set of images. Each octave has the same scale between its images, but each individual image differentiates towards the standard deviation, and each octave differs from others in the scale. In order to detect points of interest in the image, it is essential to distinguish the edges, the corners, and the blobs of the objects located inside the image. The most common way to find those points is by using the Laplacian filter. However, the application of the filter will increase the computational complexity of the algorithm since it will be applied repeatedly in each image. An alternative way to achieve the same results while avoiding the increased complexity is the creation of the Difference of Gaussians (DoG). In this instance Gaussian filter was applied to the image with accretive standard deviation, consequently subtracting the results of the filtering, which results in a new image that contains only the edges. The Gaussian filter is a smoothing filter, and this may create the assumption that other smoothing filters could be used instead in order to achieve similar results. Such an assumption is inaccurate according to research made by Koenderink [5] and Lindberg [6], who proved that, under some reasonable assumptions, the only scale-space kernel is the Gaussian function.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

where the L function is called a Gaussian scale space, * stands for convolution and $G(x, y, \sigma)$ corresponds to 2D Gaussian filter

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

Based on Lowe study [1], the difference of Gaussian function convolved with the image, $D(x, y, \sigma)$, is an efficient way to detect keypoints in scale space, and the way to compute the function D is through the difference of two consecutive scales, from the same octave, that are separated by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma) \big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

Since the L function will be computed either way, due to it being needed for the feature descriptor, we can state that using the D function is particularly efficient and the only operation needed to acquire the DoG is image subtraction.

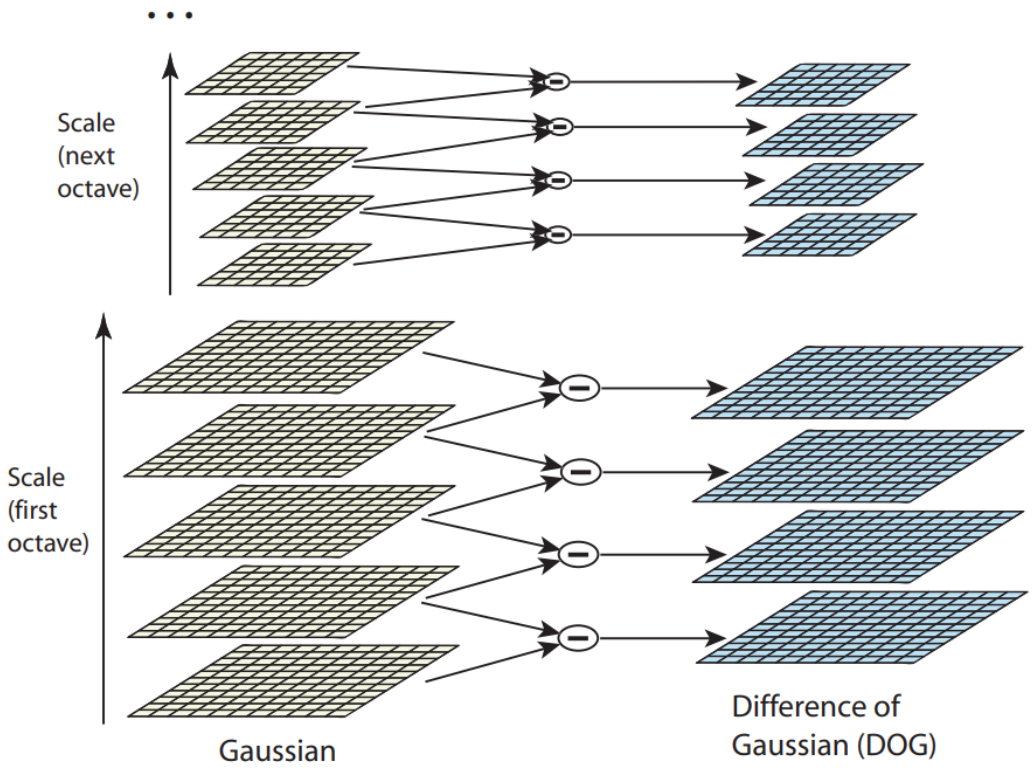


Figure 1 : The concept of the Gaussian Pyramid along with the way of creating the Difference of Gaussian Pyramid (DOG). Each octave consists of a group of images, with all of them having the same scale. The initial image of each octave is convolved repeatedly with Gaussians, resulting in the pyramid like shape on left of the image. For every new octave to be created the image is down-sampled by the factor of 2. Adjacent Gaussian images are then subtracted from each other to produce the difference of Gaussians, which again will resemble a pyramid, shown on the right of the image.
Source : Adapted from [4]

As mentioned before, the difference of Gaussian function gives results that are a close approximation to the scale-normalized Laplacian of gaussian, $\sigma^2\nabla^2 G$, as proven by Linderberg [2]. As mentioned before, it is important to detect features that are scale-space extrema. As for the scale part, it is needed to locate the stable features of the image, i.e., the features that remain interesting across all scales. Lindenberg's [2] study showed that normalization of the Laplacian with the factor σ^2 is needed for true scale invariance. In addition, Mikolajczyk [3] found that the maxima and minima of $\sigma^2\nabla^2 G$ produce the most stable image features compared to a range of other possible image

functions, such as the gradient, Hessian, or Harris corner function.

The D function is related with $\sigma^2\nabla^2 G$ through the heat diffusion equation, with the key difference of being parametrized using the standard deviation instead of the usual $t = s^2$.

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G \Leftrightarrow$$

$$\Leftrightarrow \sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(s, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}.$$

and thus,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G.$$

In essence, the term $\nabla^2 G$ can be computed from the finite difference approximation to $\frac{\partial G}{\partial \sigma}$, using the difference of nearby scales at kσ and σ. This shows that when the DoG function has scales differing by a constant factor, it already incorporates the σ2 scale normalization required for the scale-invariant Laplacian.

The factor (k − 1) in the equation is a constant over all scales and therefore does not influence extrema location. The closer the k factor approaches 1, the approximation error tends to become 0. According to D. Lowe [4], through experimentation, the value of k where the approximation error has minimum impact on the stability of the extrema points across scales is $k = \sqrt{2}$. For the current thesis, no experiment occurred in order to confirm the validity of the results described in the paper since we aim to apply the SIFT algorithm to a different application, not explicitly improve it.

As described before, an efficient way to create the Gaussian pyramid is shown in Figure 1. The initial image is repeatedly convolved with Gaussian kernels with increasing standard deviation, so images are produced that are separated by a constant k in scale space. Each octave is divided into an integer number s (every step of the octave doubles the σ), of intervals, in order to have $k = 2^{1/s}$. So, the required numbers of blurred images to be produced per stack (step) of each octave, in order for the final extrema detection to cover a full octave. The next step is the subtraction of adjacent images to produce the DoG. Upon completing the aforementioned process, follows the resampling of the Gaussian image that has twice the initial value of the σ by taking

every second pixel in each row and column. The accuracy of sampling relative to σ does not differ in comparison to that of the start of the previous octave, thus achieving rescale of the image along with a sort of upscale.

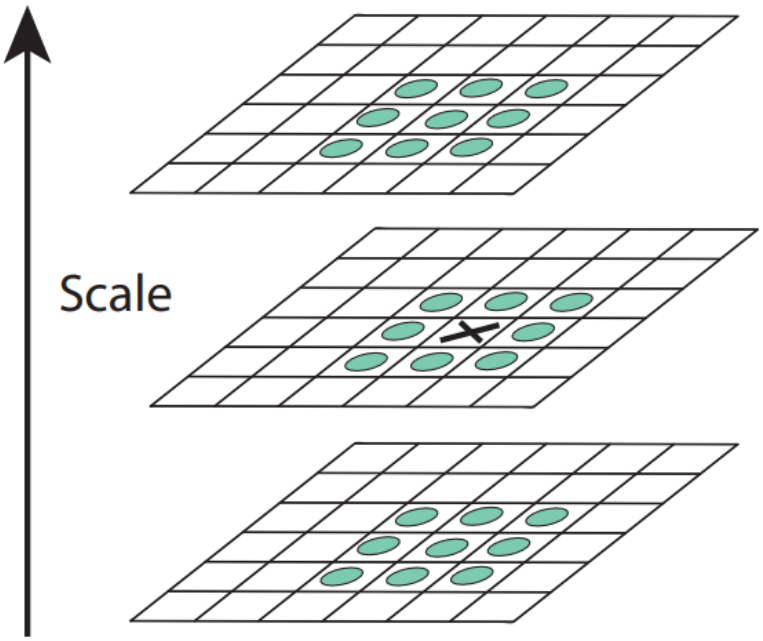


Figure 2 : The extrema points of the image are located by comparing *a* pixel (marked with x), with each of the 26 neighbouring pixels in a 3x3 region surrounding the pixel including the adjacent scales. Thus, including the parameter of scale.
Source : Adapted from [4]

Having created the Difference of Gaussian pyramid, the following task is to compare each pixel of the image with its neighboring pixel in order to determine the extrema points, in other words, to compare each pixel with the neighbouring eight that belong on the same image, along with the neighboring 18 pixels belonging to two closest scales of the octave (9 in the scale above, and 9 in the scale below). Now the problem, using this method is the repeatability, which in essence is the need to reliably locate the extrema each time the program runs. The problem occurs due to the minimum spacing between close extrema points being non-existent. For instance, a white circle on a black background will have a single scale-space maximum point, where an

elongated ellipse will possess two, one near each of the ends. So, depending on the size (elongation) of the ellipse, the two points of interest can come arbitrarily close. According to D.Lowe's study [4], through experimentations, one part for solving this problem lies with choosing the appropriate parameters during the creation of the Gaussian octaves, and in extend to the Difference of the Gaussian's octaves.

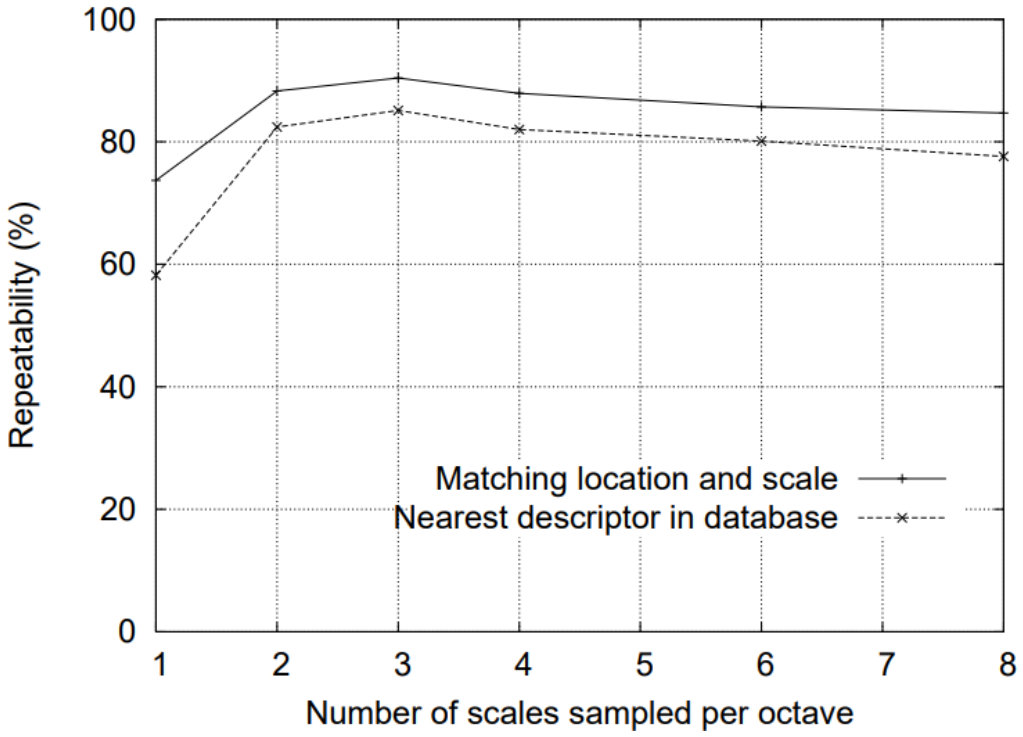


Figure 3 : The top line corresponds to the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The lower line shows the percent of keypoints that have their descriptors correctly matched to a large database.
Source : Adapted from [4]

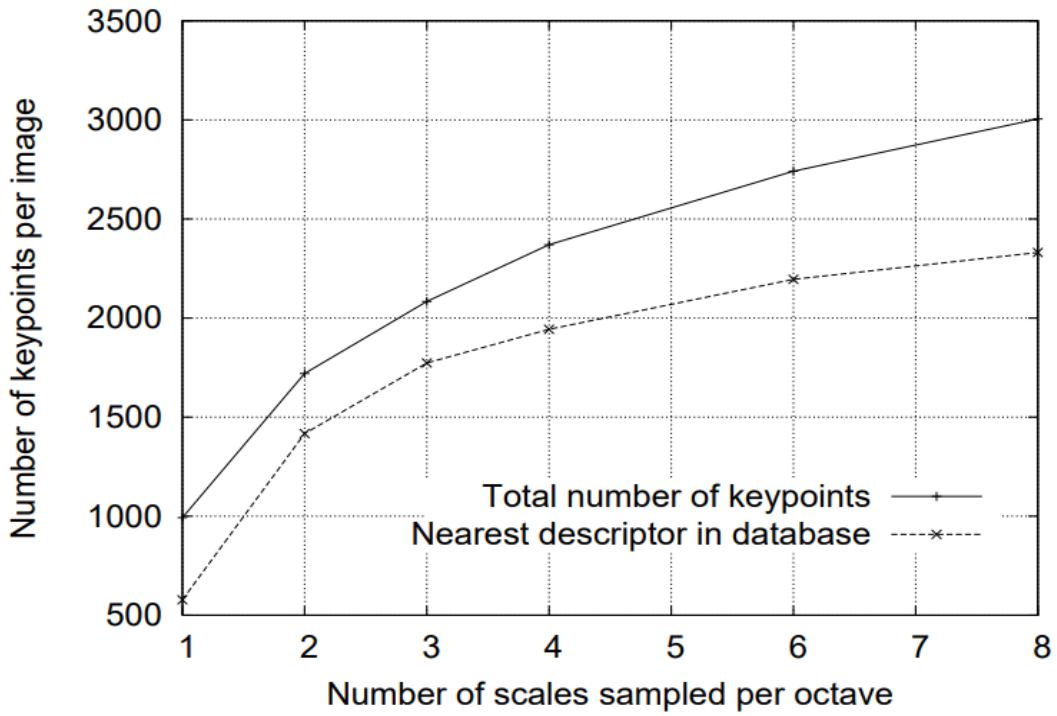


Figure 4 : This graph shows the total number of keypoints detected in a typical image as a function of the number of scale samples.
Source : Adapted from [4]

According to that study, the experiment commenced using 32 images, where they were submitted to a range of transformations, like rotation, scaling, affine stretch, brightness changes, and noise insertion. All this procedure of tempering with the images occurred using synthetic changes so that the results could be predicted. This, of course, means that using new images, where all the parameters of rotation, brightness, etc., are not targeted, the optimal values for the parameters could possibly deviate from the results of D.Lowe's results [4]. Figure 3 shows the optimal number of scales per octave to achieve repeatability, which means finding the most stable features in the image. While Figure 4 shows how the number of keypoints is fluctuating according to the number of scales. In this case, the bigger the number of scales, the number of keypoints

detected grows along with them. But the effectiveness of those keypoints is diminished since it locates not only the most stable keypoints, which are mostly disregarded during the matching process. The drawback of increasing the number of scales for the creation of the Gaussian pyramid is the computational cost due to the large number of convolutions and rescaling of the images. They are leading to the conclusion of 3 scales per octave as the optimal number. Just as the number of scales was determined through experiments, so is the smoothing rate that needs to be applied to the images to build the Gaussian pyramid.

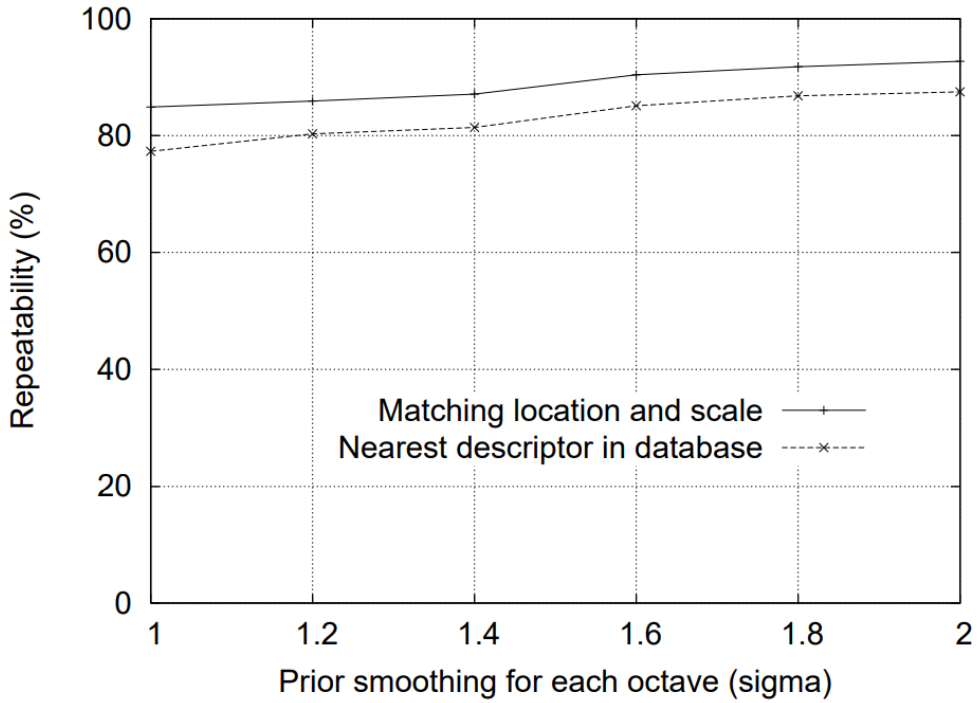


Figure 5 : The top line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The lower line shows the percent of descriptors correctly matched against a large database.
Source : Adapted from [4]

Figure 5 shows the amount of prior smoothing, which is the standard deviation of the Gaussian filter to be applied (σ), that is applied to each image level, in order to

construct the scale-space representation for an octave. According to the graph, as the σ factor increases, so does the repeatability, but due to the that comes with using large standard deviation, in terms of efficiency, $\sigma = 1.6$ is chosen as optimal since the repeatability approaches the upper limit. It is also important to note that by pre-smoothing the image prior to the extrema detection sequence, the high spatial frequencies are cut off, and to counteract this, it is possible to enlarge the size of the image, creating more samples than the original. This can happen by doubling the size of the image using linear interpolation, by assuming that the original image blur is at least $\sigma = 0.5$, and thus the new image will have $\sigma = 1$.

2.2.2 Feature point localization

Up to this point, it has been explained how the basic keypoint detector operates, but there is still the problem of detecting even more accurately the extrema points of the image since the points located until now are coarsely localized, at best to the nearest pixel. The next step is to perform a more detailed localization to the point of sub-pixel level of accuracy while removing poor features. This procedure is a method developed by Brown (Brown and Lowe, 2002) [5], whereby fitting a Taylor expansion to fit a 3D quadratic surface (in x, y, and σ) to the local sample points to determine the interpolated location of the maxima and minima. The expansion, ignoring terms above the quadratic of the scale-space function $D(x, y, \sigma)$, shifted to the proposed point:

$$D(z_0 + z) = D(z_0) + \left(\frac{\partial D}{\partial z}|_{z_0}\right)^T z + \frac{1}{2} z^T \left(\frac{\partial^2 D}{\partial z^2}|_{z_0}\right) z \qquad (1)$$

where the derivatives are evaluated at the suggested point $z_0 = [x_0, y_0, \sigma_0]^T$ and $z = [\delta x, \delta y, \delta\sigma]^T$ is the offset from that point. The location of the extrema $\hat{z}$ is calculated by setting the derivative with respect to z equal to zero, and the equation becomes:

$$\hat{z} = -\left(\frac{\partial^2 D}{\partial z^2}|_{z_0}\right)^{-1} \left(\frac{\partial D}{\partial z}|_{z_0}\right) \qquad (2)$$

The derivatives of D are calculated using the differences of neighboring pixels around the target point. The result is a 3×3 linear system that does not burden the algorithm with excessive computational cost. In case the offset value that determines where the extrema point is greater than 0.5 (half a pixel), the procedure needs to repeat relocating the target pixel since the actual maximum will closer to one of the neighboring pixels. Now, values that are calculated at that new extremum point and do not satisfy a certain threshold are discarded as they are too sensitive to noise. The equation to find those values is:

$$D(\hat{x}, \hat{y}, \hat{\sigma}) = \Delta(z_0 + \hat{z}) \approx D(z_0) + \frac{1}{2}\left(\frac{\partial D}{\partial z}|_{z_0}\right)^2 \hat{z}$$

Another step to take towards finding the most stable keypoints, is discarding keypoints that were found along an edge in the image. This is important since those points have a large principal curvature across the ridge (edge), but a low one along the other direction, making that point relatively unstable towards the position on one of the axis, while a well-defined peak that has high curvature in both directions does not insert ambiguity to the exact location of the extremum. The principal curvature can be estimated from a 2×2 Hessian matrix, H, computed at the location and scale of the keypoint:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

where the derivatives can be calculated using the differences of neighboring pixels. Since only the eigenvalues ratio is needed, there is no need to compute them. Assuming the eigenvalue with the greatest magnitude is represented by k_1 and the one with the smaller magnitude by k_2, the sum of the eigenvalues can be determined by the trace of the H, while the product by the determinant, resulting in the following equations:

$$Tr(H)^2 = D_{xx} + D_{yy} = k_1 + k_2,$$

$$Det(H) = D_{xx}D_{yy} - D_{xx}^2 = k_1 k_2.$$

$$\frac{Tr(H)^2}{Det(H)} = \frac{(k_1 + k_2)^2}{k_1 k_2} = \frac{(rk_2 + k_2)^2}{rk_2^2} = \frac{(r+1)^2}{r}$$

where r corresponds to the ratio between the eigenvalues so that $k_1 = rk_2$. In the rare case where the Det(H) is negative, then that means that the point under inspection is not an extremum since the curvatures have different signs and therefore is disregarded. From the equation that describes r, it is easy to realize the minimum number curvature will be achieved ($min\ (\frac{(r+1)^2}{r})$), when the two eigenvalues have equal magnitude, and the greater the ratio r is, the greater the curvature will get. So, in order to check if the principal curvature is below a threshold, the following need to be satisfied:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

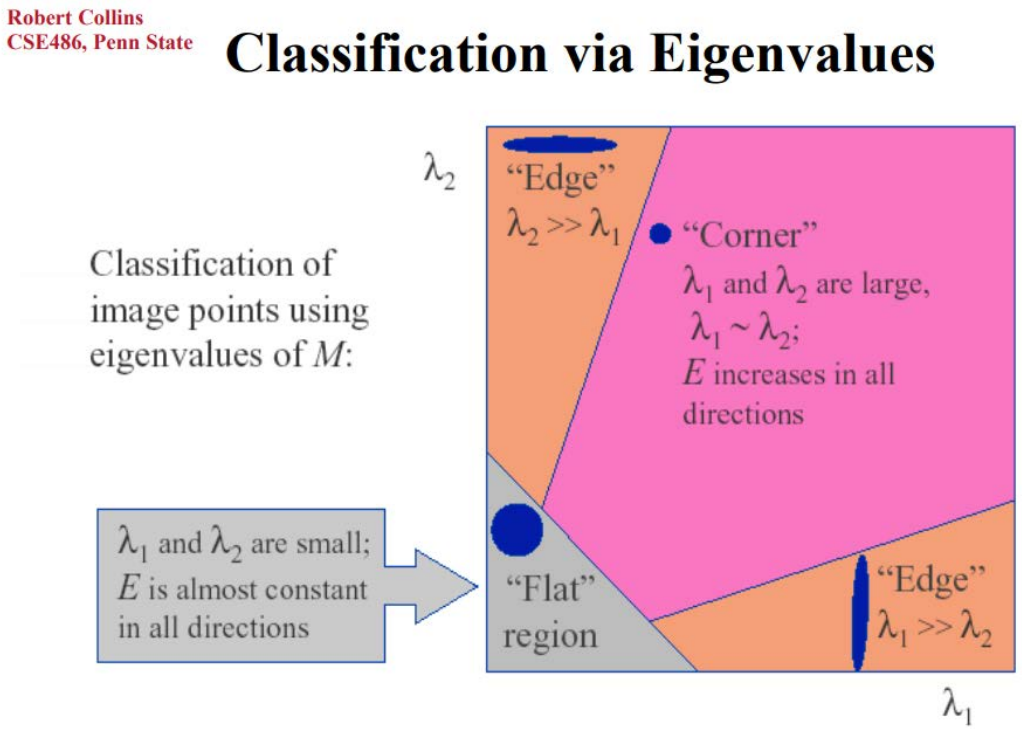


Figure 6 : Using the eigenvalues it is possible to determine if the region around a keypoint contains an edge or even corner. There is also no need to calculate the individual eigenvalues, but only the ratio suffices to discriminate the most important keypoints.

2.2.3 Orientation of keypoints

Finally, the keypoints at which the extrema of the image have been located and distinguished are the most stable ones. However, to reliably use all those collected points, it is not enough to know just the exact location in the scale-space. In order to achieve rotation invariance between two images (up to a limit of rotation), besides the location of the keypoint, some sort of information about the direction of the image is paramount, otherwise in the matching process, the final matched image may have a wrong rotation which while may cause the severe image distortion in case of blending the two. For example, an image with some feature inside that matches another in a new image, which has a slight rotation, may match, and the rotation on the immediate surrounding area of that feature could be minimal, to the point of having one-to-two-pixel offset, but may have consideranble impact to the background information of the images.

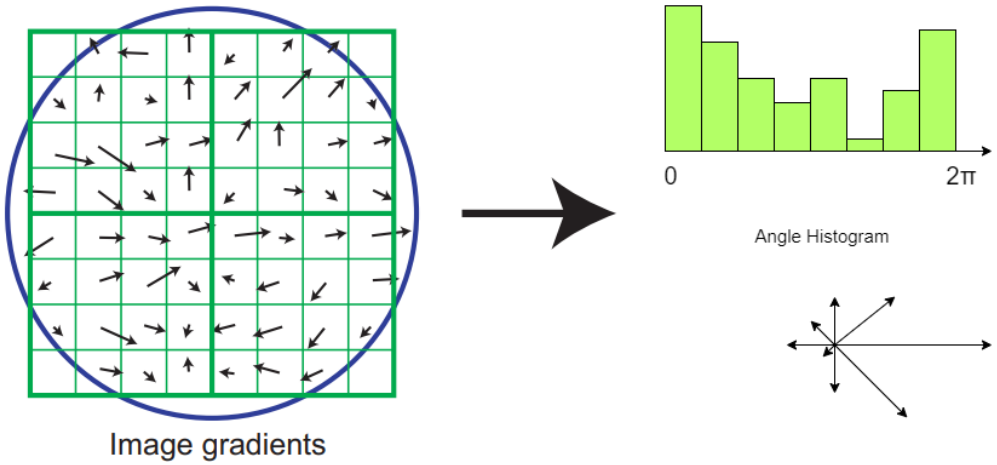


Figure 7 : On the left, there is a sample of the image where at the centre is a keypoint. The magnitude and orientation of the gradient are computed and represented with the vectors. After smoothing using a Gaussian filter (blue circle) an angle histogram is created that will help to filter out some keypoints.

The procedure of assigning orientation to each keypoint starts by selecting the scale of the keypoint and use that image, which is a Gaussian smoothed version of the original L, so that all computation to be performed corresponds to the correct scale and thus be scale-invariant. Then, for every pixel in the image, $L(x, y)$, the magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are to be calculated using pixel differences:

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + \big(L(x,y+1) - L(x,y-1)\big)^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

With all those magnitudes and orientations be precomputed, an orientation histogram can be created based on the gradient orientation of each pixel that within a certain range around every keypoint. The histogram possesses 36 bins representing the 360-degree range of orientations, with a resolution of 10 degrees per pixel, which is divided uniformly across the 36 bins. Now each pixel that is added to the histogram is weighted corresponding to the magnitude of the gradient and by a Gaussian weighted circular window with σ that is 1.5 times the scale of the keypoint. The weight system is used in order to keep track of the dominant directions of local gradients, which will appear at the peaks in the histogram. The biggest peak of the histogram is selected along with any other local peak that is within 80% of the value of the highest peak in order to create a profile of the certain keypoint. Also, parabola interpolation is used to increase the accuracy of the process by targeting the three highest values of each peak. This process also contributes to the stability of each keypoint.

2.2.4 Feature Descriptor

Finally, all the procedures to describe with accuracy the location, scale, and orientation of the keypoints of the image are completed. All these parameters will allow for a local 2D coordinate system to be created, one that describes every keypoint and its surrounding area. The next step is to construct a descriptor that holds enough

information about those areas neighboring each keypoint, that is highly distinctive so that there will be as few mismatches as possible, while at the same time be as invariant as possible to remaining variations, such as those of change in illumination in the image, as well to an extent, tolerance to affine transformations, i.e., changes in the 3D viewpoint. One way to achieve some of those effects would be by using the intensities of the pixel in a certain area circling the keypoint and always considering the scale in order to match them through a normalized correlation measure. The version of the descriptor that was finally implemented is based on the usage of the gradient magnitude and orientation. Having precomputed the magnitude and orientation of the pixel in the desired area and taking account the Gaussian blur to achieve scale invariance, the next problem is the orientation invariance that is needed to be achieved. For that, all the gradients and orientations need to be corrected (rotated) relative to the orientation of the keypoint. The further a pixel is from the location of the keypoint, the lesser is the importance of that particular pixel, so there should be a weighting system in order to avoid random high values of magnitude in the outer circle inside the designated area surrounding the keypoint. That value may often be high, not due to some kind of noise or misregistration error, but may be related to another neighboring extremum (keypoint) and thus have a significant magnitude. Either way, such occasions can and should be foreseen, and countermeasures need to be taken. This leads to the usage of a Gaussian weighting function with σ chosen so that it is 1.5 times the width of the descriptor window, and of course the weight diminishes smoothly the furthest from the center.

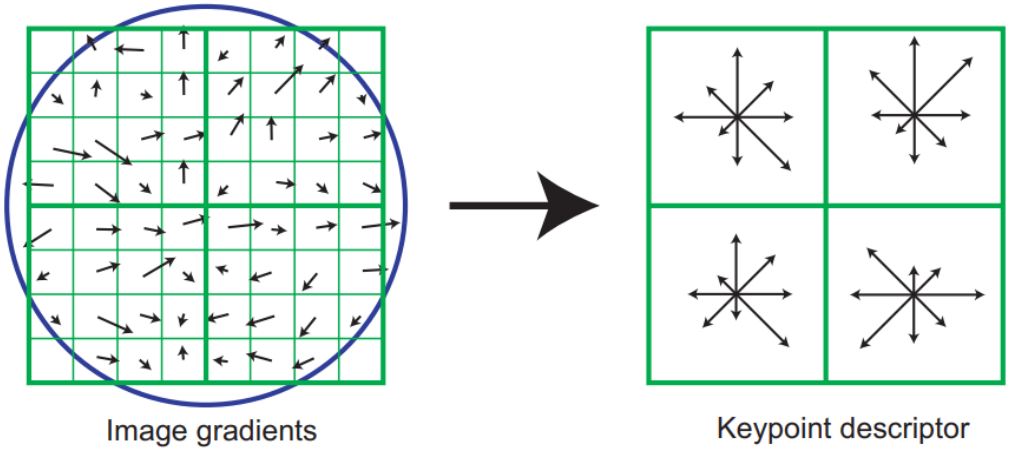


Figure 8 : On the left is a part of the image containing a keypoint. Every pixel's gradient has been calculated and is represented with an arrow. On the right is the keypoint descriptor that contains an orientation vector that describes the corresponding subregion in the image.

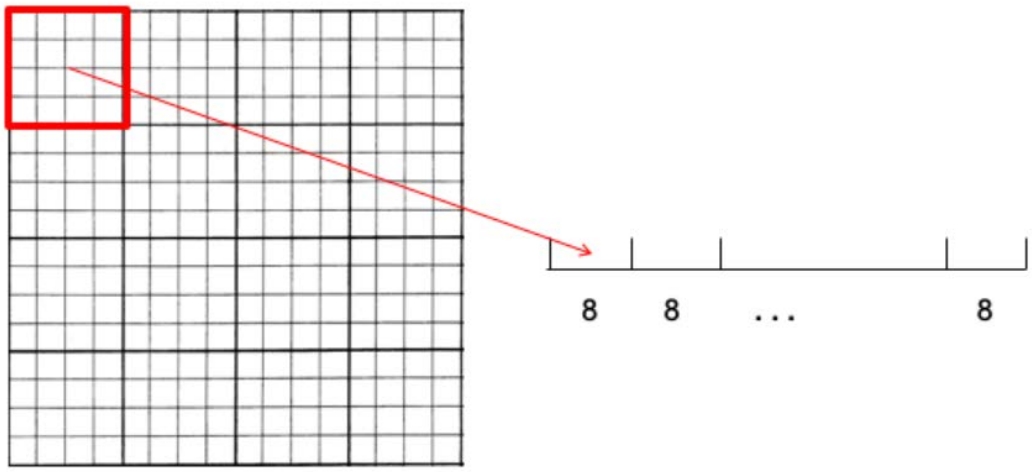


Figure 9 : On the left, is the 4x4 area surrounding the keypoint that needs to be analysed. This area is further divided into small subregions and an orientation histogram is created for each one.

Next up is the division of the selected area around the keypoint into 4×4 subregions where a histogram for each region is created. The way that is represented in Figure 8, each arrow indicates a direction that is represented by a bin in the histogram (each bin corresponds to different orientations), and the length of each arrow is proportional to the cumulative magnitude of all the vectors of the gradients that point to the same direction and belong to one of those subregions. The descriptor in Figure 9 is the descriptor implemented in the algorithm and consists of a normalized 128-dimensional vector, a 4×4 spatial grid (subregions) is used, and each subregions histogram divided into 8 orientations (128 = 4×4×8).

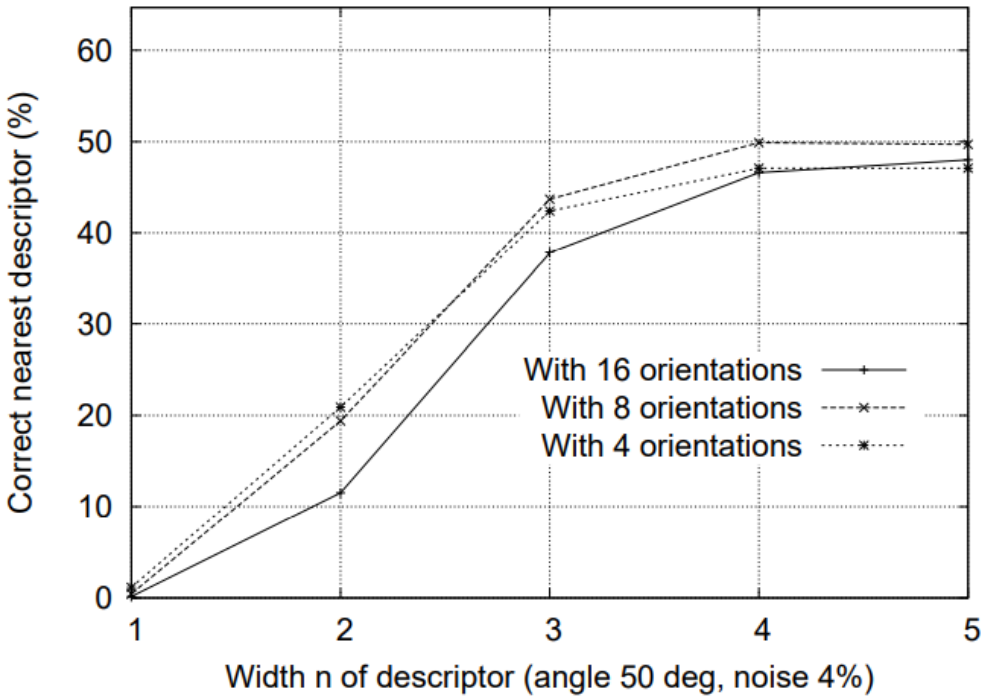


Figure 10 : According to this graph the optimum width of the descriptor, since above 4 there is no significant change, along with the most suitable number of orientations to be used.
Source : Adapted from [4]

For simplicity, in order to understand better the purpose and the way the descriptor operates, an alternative version such as Figure 8 will be used as an example, where

there is only a 2×2 descriptor array computed by an 8×8 set of samples. This style of 4×4 sample regions provides a significant shift in gradient positions. A gradient sample on the left can sift up to 4 sample positions while still contributing on the right, and thus giving tolerance to larger local positional changes. A problem that can occur in this stage is boundary effects, where the descriptor abruptly changes because of a sample shifting from being in one histogram and changes to a neighboring histogram or an orientation change. To counter this problem, trilinear interpolation is incorporated in order to distribute the value of each gradient weight of 1- d for each dimension, where d is the distance of the sample from the central value of the bin as measured in units of the histogram bin spacing. The vector that is created and represents the descriptor can be modified to counter some of the unwanted illumination changes that can occur between images. In the case of linear changes, such as a simple offset between the values of luminance, either an increase or decrease of brightness, the problem is already solved since the gradient values are computed by pixel differences. As for contrast changes in the image, those will multiply the gradients of the pixels by a constant, in which case a vector normalization will suffice as a countermeasure. With all these problems solved, it can be said that the descriptor is invariant to affine changes in illumination. As for the non-linear changes of illumination, the solution is not as clear. Such changes that occur due to camera saturation and other sources will have a considerable effect on the vector magnitudes for some gradients but are less likely to affect the orientation also. To reduce this effect, a threshold is employed to the already normalized gradient magnitudes so that no value exceeds the 0.2 threshold limit and then renormalizing. This will make the system value the importance of the large gradients less and make it focused on the distribution of the orientations. The value of 0.2 has been suggested by D. Lowe in his paper [4].

2.3 MATCHING FEATURES

Like the feature detector algorithm, there are numerous approaches for the problem of matching two images. One such approach is to utilize the data of the pixels of the images at the connection point by shifting and wrapping them relative to each other and then check if the corresponding pixels much. Such methods are often referred to

as “direct methods” and require the utilization of an error metric in order to decide the accuracy of the matching of the two images, along with a suitable search method, to find the pixels that minimize those error metrices. Such methods include the full search, which is an exhaustive search of the pixels of the two images, searching for all possible alignments. The full search has a major drawback that is the time needed to complete the search due to the high computational complexity that comes with checking all possible pairs for matches, making this process unsuitable for many applications. Alternatively, in order to avoid such time-consuming methods, Fourier transform can be used to speed up the computation. Also, a Taylor series expansion is often used to increase the accuracy of the methods. As for the error metrics, there are also a variety of techniques. The simplest method, after aligning and sifting the images relative to each other, is to find the sum of the squared differences (SSD) function. Given the first image $I_0(x)$ sampled at a pixel location $x_i = (x_i, y_i)$, and the goal is to find the corresponding pixel on the second image $I_1(x)$, then the function that calculates the SSD forms as:

$$E_{SSD}(u) = \sum_i [I_1(x_i + u) - I_0(x_i)]^2 = \sum_i e_i^2$$

where u corresponds to the displacement of the two images and $e_i = I_1(x_i + u) - I_0(x_i)$ is the residual error. An alternative method is replacing the squares error terms with a robust function $p(e_i)$ to achieve

$$E(u) = \sum_i p(e_i).$$

This is known as robust error metric, and a widely used robust function is the sum of absolute differences (SAD) which, compared to the SSD, is a function that does not grow as quickly due to the lack of the squares. Both of the above metrics are not suitable for gradient descent approaches since the function are not differentiable at the origin. Another similar metric is that of the spatially varying weights, which is similar to the SSD metric but has the ability to be applied only in certain parts of the image. It is also known as windowed SSD function. This, of course, gives the advantage of filtering the pixel that is needed to be checked between the images, assuming the knowledge of the limits of the overlap region that the images will have.

$$E_{WSSD}(u) = \sum_i w_0(x) w_1(x_i + u)[I_1(x_i + u) - I_0(x_i)]^2.$$

Up to this point, a number of techniques have been mentioned that are used for image alignment (matching). All of those techniques compose a small number of the existing methods developed for the solution of the problem of the matching images, with each one having advantages over the others and vice versa. These methods are primarily designed and used for intensity base stitching, where the matching of the images will be based on the information given by the values contained in the pixels but not the features of the images described by the pixels in that area. In this thesis, a feature-based technique is used (SIFT algorithm), which is another approach to solving the same problem, so all of the above methods cannot be used directly since it would waste all the resources that were used to accurately describe and locate the keypoints of each image. Although that does not completely exclude the use of those techniques or some of the elements they use.

Feature matching is essentially the procedure in which, utilizing the feature detectors that are already produced by the SIFT analysis in order to find the correlation between the features, it is possible to locate identical features that are located in different images so that the two images using the proper displacement can be matched correctly. Image stitching is widely used in a variety of applications, including document mosaicing, video stitching, medical imaging, and others. Depending on the constraints created by the different applications and their needs, entirely different approaches can be adopted to accomplish the stitching process. In instances where a large collection of images needs to be stitched into one image and the geometric correlation of the images is obscure, the most common approach would be the analysis of the whole collection, finding keypoints in every image and afterwards by comparing those features the exact location of each image is uncovered. On the other hand, when it comes to video stitching, an interesting technique is one called “detect then track,” where at first the translation between matching points of neighboring frames is calculated and then using that distance to predict the next displacement of that feature in the next batch of frames. A major advantage of this procedure is the need to analyze the images infrequently, mostly when tracking has failed. Another constraint to be found in some of the

stitching applications is affine translation. Especially in the panorama photography mode, that almost every modern phone possesses, the user takes consecutive pictures while primarily standing at the same location while rotating around himself. Those pictures taken with this method do have not only a displacement on the x and possibly the y-axis but also possess a translation on the z-axis that has to be considered when matching the keypoints and finally blending the images together. The problem of in-plane rotations is restricted to finding the primary orientation that characterizes the feature before computing the descriptor of that particular keypoint. One way of finding that dominant orientation is searching for the average gradient orientation in a designated area around each feature. Another more promising approach is the one adopted by Lowe [4], which searches for the maximum value of the gradient in the orientation histogram. This method is generally more accurate than using the average orientation. Since the application of this thesis is based on the microscope, not all of the above problems are present, and thus not all the countermeasures for them need to be taken in order to accomplish the goal that is image stitching. The method that was used for this application resembles mostly the one that is deployed when there is a large collection of images to be stitched together and create a big mosaic. The main difference though, is that since it is possible to acquire the images one by one, there is no need to analyze all of them at once.

For the feature matching to take place, there is a need to find matching pairs of features between the two images in question. The simplest and most accurate yet most time and resource-consuming method is that of the exhaustive search in which every feature that is contained in each image needs to be compared to every feature belonging to the second image. This method has great results regarding finding a matching pair of features whose descriptors have minimal differences, but regarding its computational complexity, which is $O(n^2)$, where n represents the number of features. The high computational cost is making the usage of this method forbidding for some applications where the completion speed is important, or even for applications that the images contain a large number of features.

2.3.2 RANSAC

In these types of situations, the most widely used method to find matching features between images is the Random Sample Consensus (RANSAC) [6]. This algorithm is a relatively simple yet effective way to filter data contaminated with outliers. Generally, in data sets, the data that are used can be divided into two categories, inliers, and outliers. Outlier is defined as an observation that deviates too much from other observations that it arouses suspicions that it was generated by a different mechanism from other observations [7]. Inlier, on the other hand, is defined as an observation that is explained by the underlying probability density function. In clustering, outliers are considered as noise observations that should be removed in order to make more reliable clustering [7]. RANSAC is a highly effective technique of grouping and distinguishing data into those categories, and the procedure can be implemented in three primary steps. For example, when fitting a line in a set of data, which is also represented in Figure 11, the first step would be to choose two random samples, which at first will be considered inliers, even though that may not be true in the final result. According to those selected inliers, a model will be constructed. In this example, a line will be fitted. The second step of the algorithm would be to count the number of data points that will agree with the constructed model, and those will be the points that their distance from the line created won't exceed a threshold that was chosen. The last step for RANSAC to be completed is to repeat this process again, choosing a new set of inliers each time so that the iteration with the most inliers will be considered correct. The stopping criterion for the algorithm may vary depending on the application and the desired results since the accuracy of the model will increase along with the number of iterations. Furthermore, the iterations N needed can be calculated in order to achieve the desired accuracy.

$$N = \frac{\log(1-\rho)}{\log(1-r^s)} \tag{2.3.1}$$

Where p stands for the probability of finding a model without any outliers, r is the inlier ratio of the data set, while s represents the number of data points that are assumed as inliers (in the current example 2).

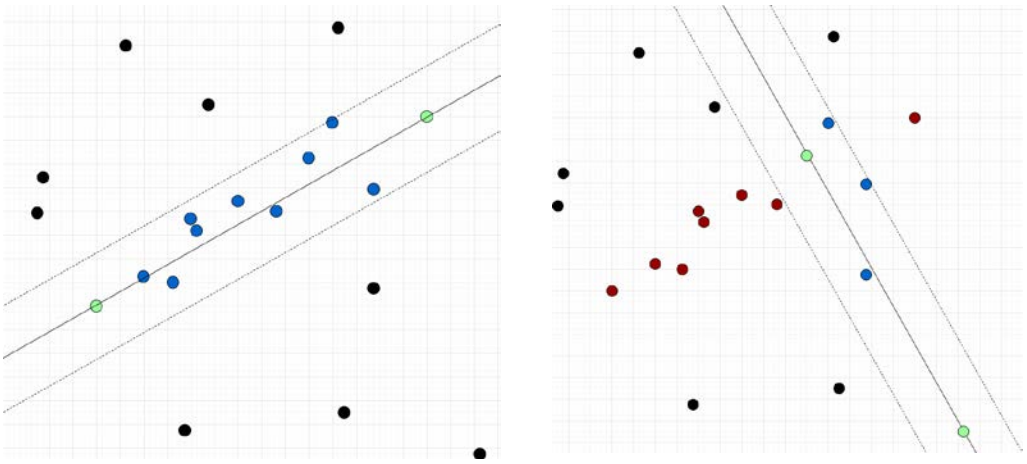


Figure 11 : Left: First iteration of RANSAC, two points are chosen as temporary inliers for the current run that will the define a line. All the points (blue) that are within the acceptable range (dotted lines) will be considered as inliers and the black ones will

According to a study by Fischler and Bolles [6] The RANSAC paradigm is more formally stated as follows:

Given a model that requires a minimum of n data points to instantiate its free parameters and a set of data points P such that the number of points in P is greater than n [$\#(P) \geq n$], randomly select a subset SI of n data points from P and instantiate the model. Use the instantiated model M1 to determine the subset SI* of points in P that are within some error tolerance of Ml. The set SI* is called the consensus set of S1. If # (SI*) is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use SI* to compute (possibly using least squares) a new model MI *. If # (SI*) is less than t, randomly select a new subset $2 and repeat the above process. If, after some predetermined number of trials, no consensus set with t or more members have been found, either solve the model with the largest consensus set found or terminate in failure [6].

2.4 BLENDING

The last task needed for the whole sequence of stitching to complete is the blending process of the images. Similarly, to all the processes that were mentioned up to this point, depending on each situation, a different approach is needed to accomplish the task of blending. The first thing that needs to be cleared is the final composite surface along with the view of the reference image since in each case there will be a need of some parametrization for the coordinates assuming the final surface is not flat. For example, for a simple panorama shot taken by a user, it is safe to assume that the camera will be rotated around the z-axis (as shown in Figure 12) to take the consecutive shots instead of moving along the x-axis for the additional image. When such a thing occurs, while not having a large number of images to stitch, it is possible to disregard the distortion created by the rotation around the z-axis by warping all the images to the coordinate system of the reference image, which should be the image placed at the geometrical center of the final result, not necessarily the first image that was taken. However, when the rotation around the z-axis is substantial, such an approach will produce undesirable distortion to the contents and the connection of the images. (In practice, flat panoramas start to look severely distorted once the field of view exceeds 90◦ or so.) The usual choice for compositing larger panoramas is to use a cylindrical [8], [9], or spherical [10] projection [11]. Having a projection other than flat requires the parametrization of the pixel coordinates and the construction of the mappings between the input and output image. When the final compositing surface is a texture-mapped polyhedron, a slightly more sophisticated algorithm must be used. Not only do the 3D and texture map coordinates have to be properly handled, but a small amount of overdraw outside of the triangle footprints in the texture map is necessary to ensure that the texture pixels being interpolated during 3D rendering have valid values [11].

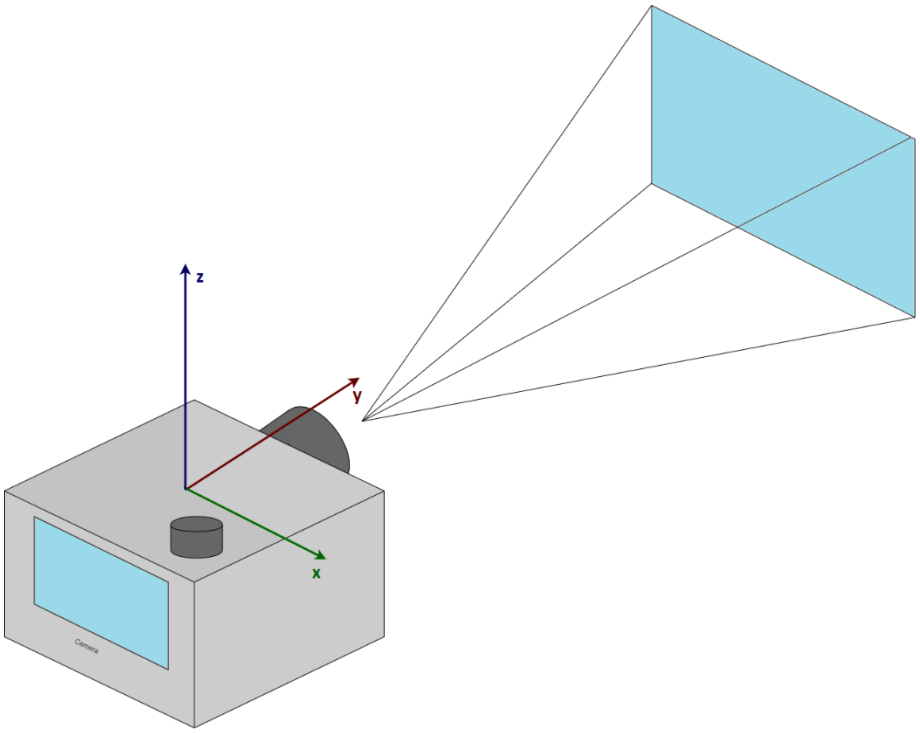


Figure 12 : The axes of the camera

By clarifying the coordinate system that will be used according to the needs of the application, it is possible to move on to the next task, which is selecting the appropriate algorithm for blending the images. Generally, the target of such algorithms, aside from stitching the images, is to minimize as much as possible and eliminate certain effects the are being produced in the stitching process, like visible seams at the connection point, which appears due to exposure difference of the two images, blur which is caused by misregistration of the pixels and ghosting that appears when there is a moving object.

The most straight forward algorithm to use is to calculate the average of each pixel to create the final composite image.

$$C(x) = \frac{\sum_k w_k(x)\, I'_k(x)}{\sum_k w_k(x)}$$

I'_k stands for the composite image, and w_k takes the value of 1 at a valid pixel and 0 everywhere else, i.e., it is going to affect pixels of the warped image that are part of both the two basic images. Although this method is the simplest to use, little work is done towards eliminating the effects of visible seams, blurring, and ghosting. An improvement over this method is called feathering, which differs from the simple averaging by alternating the distribution of the weight so that they will be focused more on the center of the image and fade away while moving to the edges. This can be done by computing a distance map,

$$w_k(x) = \left\| argmin_y \{ \|yy\| \mid I'_k(x+y) \; is \; invalid \} \right\|$$

where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel [11]. This approach reduces substantially the visible seams, however does not solve the blurring and ghosting effects.

An alternative method for blending is the one designed by Burt and Adelson [12], which involves the use of the Laplacian pyramid. According to the algorithm, each warped image is converted into a band-pass (Laplacian) pyramid, which involves smoothing each level with a 1/16(1,4,6,4,1) binomial kernel subsampling the smoothed image by a factor of 2 and subtracting the reconstructed (low-pass) image from the original. This creates a reversible, overcomplete representation of the image signal. Invalid and edge pixels are filled with neighboring values to make this process well defined [11]. Next, the *mask* (valid pixel) image associated with each source image is converted into a low-pass (Gaussian) pyramid. These blurred and subsampled masks become the weights used to perform a per-level feathered blend of the band-pass source images [11]. Finally, the composite image is reconstructed by interpolating and summing all of the pyramid levels

Poisson Image Editing

This is another more sophisticated solution to the blending problem than the aforementioned methods. In this case, the images to be fused are distinguished to source (S) and target (T), and there is also a mask (Ω) that corresponds to the region of the source that will be moved to the target image. Another way to describe S and Ω,

according to Perez, Gangnet, and Blake [13], S, Ω now become finite point sets defined on an infinite discrete grid. Note that S can include all the pixels of an image or only a subset of them. The boundary of Ω is now $\partial\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq 0\}$ [13]. The basic idea is to reduce the color mismatch between and target images by creating the composite image in the gradient domain. In other words, the goal is, the gradient of the composite inside the Ω region to be as close as possible to the sources image gradient while matching the boundary ($\partial\Omega$) to the target image. This is possible to achieve by solving the minimization problem of the following equation:

$$min_f \iint_{\Omega} |\nabla f - v|^2 \ with\ f|_{\partial\Omega} = f'|_{\partial\Omega}$$

Where f corresponds to an unknown scalar function defined over the interior of Ω [13], v is a vector field defined over Ω that is also called guidance field and f' is the scalar function defined over S excluding the Ω region. The solution of the equation is the unique solution of the following Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \operatorname{div} v \ over\ \Omega, with\ f|_{\partial\Omega} = f'|_{\partial\Omega}$$

where $divv = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $v = (u, v)$ [8]. The algorithm that to Perez, Gangnet and Blake [8] introduced does not end here since they also introduced an extension to the method described above that is called mixing gradients compositing where the guidance field v will not always use information from the source image ($v = \nabla g$), but will choose depending on the magnitude of the gradients of candidate images, which can also be described as:

$$\text{for all } x \in \Omega, v(x) = \begin{cases} \nabla f'(x) & \text{if } |\nabla f'(x)| > |\nabla g(x)|, \\ \nabla g(x) & \text{otherwise.} \end{cases}$$

3 Research Design

3.1 REVIEW THESIS GOAL

The goal of the current thesis is to design a program that will aid the user, who most likely is a doctor, to complete a particular procedure needed when using the microscope. Furthermore, by making the whole procedure easier along with minimizing the time needed to be dedicated and in case there is a necessity to review a specific sample, it will be possible to skip the procedure since the results of the first run of the program on a sample will be saved and easy to access. In this section, we will analyze the algorithms that were chosen among those mentioned in section 2 while examining the reasons that they fit best this particular scenario of the microscope. Before moving on to the algorithms, first of all, there is a need to establish which programs (apps) were used along with the equipment for all the tests that took place.

3.2 APPLICATIONS AND EQUIPMENT

3.2.1 Lumnia Microscope

The microscope used for the current thesis is the Lumnia motorized microscope, one that the lab developed. One important characteristic that differentiates this microscope compared to the classical one is the ability to operate it and observe the sample not through the ocular lens but rather through a screen and using the same screen to navigate the samples, instead of manually controlling the location of the stage. The prototype used in the lab has an intel core i9 9900K for CPU, 64GB DDR4 RAM, and as for the operating system, Windows 10 are installed. These components make the microscope a computer capable of running multiple demanding processes. Additionally, it is kitted with a touchscreen that will help with the visualization of the samples as well as navigate through the sample and choose which process to operate. An alternative option for navigating through the sample is the usage of a joystick that can be plugged in to provide with more instinctive control of the movements. Furthermore, the Lumnia is kitted with a multimodal camera, being able to capture images in different wavelengths, thus giving the ability to observe different

characteristics of the sample each time. Moreover, it possesses an epi-illumination system, magnification encoder, XY linear translation stage as well as a motor for the z-axis. The magnification encoder, in conjunction with the translation in the z-axis, is responsible for focusing on the sample, while the XY translation motors move the stage, so that a new image can be taken in a different location of the sample. The accuracy of the XY movement is up to 0.75μm, while the range of motion is up to $10cm^2$. Through the menu built for Lumnia, the user is able to perform some operations, one of which is the stitching. It is essential to mention that because of the XY axis resolution and the ability to know the exact location of each image by reading the movements of the motors, it was possible to translate each movement of the stage from μm to pixels, and consequently have knowledge whether a stitch is successful or not.

3.2.2 QT and OpenCV

The entirety of this thesis was implemented using the C++ programming language and was written in the Qt platform. Qt is a widget toolkit for creating multi-platform applications capable of running on most desktop platforms, and provides tools for creating GUIs. The Qt software is available in both commercial licensing as well as open-source licenses, and in this case, the latter was used. Apart from the environment used, there are quite a few libraries that were created to support C++, one of which that was extensively used in the current thesis is OpenCV (Open Source Computer Vision Library), which is an open source computer vision and machine learning software library.

3.3 DESIGN ANALYSIS

When a user operates a classical microscope, regardless of the type of microscope, either being optical, electron, etc., in other words, a non-automated one like the those mentioned earlier, after choosing the desired magnification, he scans the samples that have been placed, sometimes multiple times to arrive and examine a specific location on the sample or has to view multiple parts of the same sample. By using an image stitching program, this procedure is expected to become more accessible and faster. The image stitching procedure will result in making a panorama-

like image of the sample inserted into the microscope by taking the multiple images captured by the microscopes camera into a large high-resolution image, so the final image (output) will encapsulate the whole sample while aiming to minimize any alteration of the contents to the pixels of the original images. The image stitching procedure can be divided into three primary stages, as shown in Figure 13. In the paragraphs to follow, an analysis will commence on each stage separately.

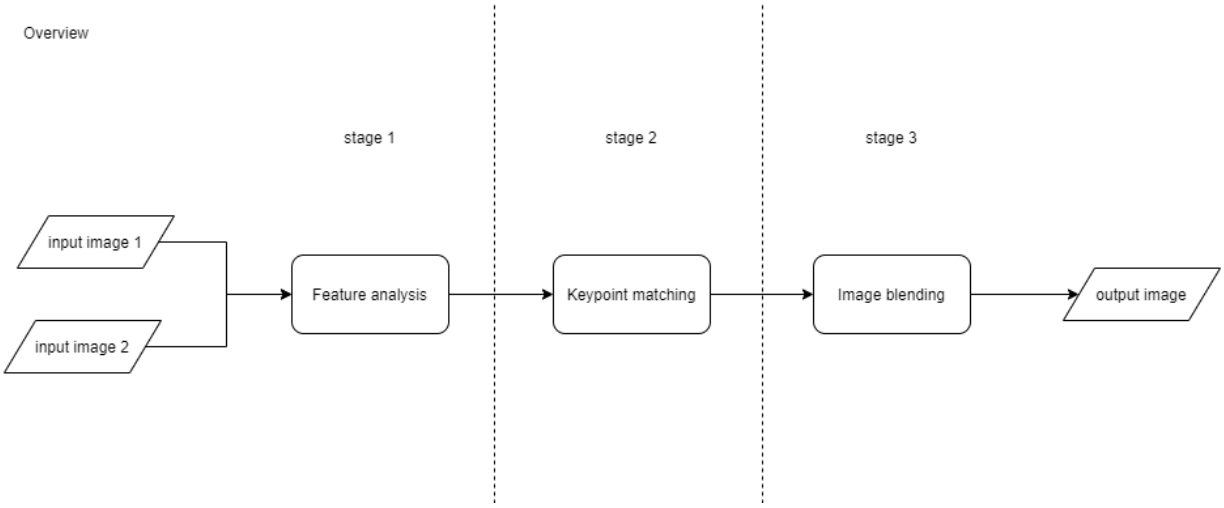


Figure 13 : Overview of the image stitching procedure that can be divided in three sections.

3.3.1 Feature analysis (Stage 1)

The first order of things is to load the images into the program in order for the analysis to start. The default assumption here is that all the images will be in an RGB format, but the algorithm is not restricted to just a 3-channel image. It can also operate without any changes, even if a 1-channel image is inserted. That is due to the fact that the inserted images are initially converted to grayscale in order to proceed with the algorithm. It should also be noted that although the feature analysis done by SIFT will commence in each image separately, and only at the next stage (matching), the comparison between the images will take place. The next step is determining the

number of octaves to be used, which will be 4, and the number of scales for each octave, in this case, are 5 scales. Now the task is to compute the Gaussian octaves. This is accomplished by blurring the image as many times as the number of scales we want in each octave, each time using a different standard deviation (σ), increasing it in each scale. Afterwards, the image is rescaled to have half the size of the images in the previous octave and repeating this sequence until all octaves are constructed. A function of OpenCV was used to apply the gaussian blur, as well as the rescaling function. Initially, a value is assigned to the standard deviation for the first scale of the base octave, $\sigma = 1.6$, and k also takes the value of $\sqrt{2}$, and for all the rest of the scales in the octave the standard deviation will be calculated from the following equation: $\sigma' = k * \sigma$. It should be noted that Lowe [4] in his paper suggests the initial standard deviation be $\sigma = 1.6$. Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) [4]. The result of this procedure can be seen in Figure 14 where each column of images corresponds to one octave, it also shows the size of the pictures in each octave. It is visible the difference between the scales due to the standard deviation of the Gaussian filters.

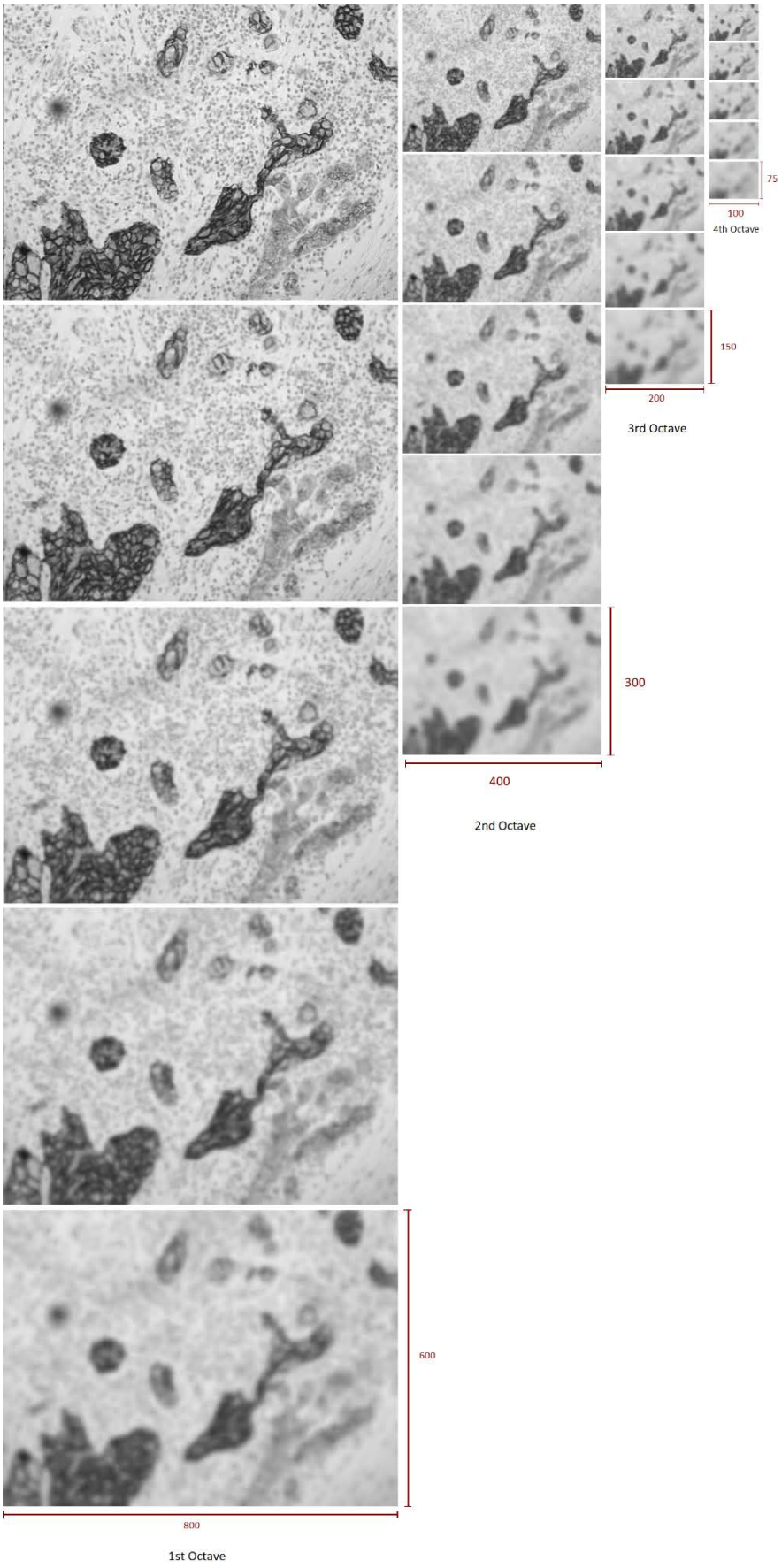


Figure 14 : An example of the Gaussian pyramid. In this case there are 4 octaves with 5 scales in each one. The red lines show the size, in terms of width and height, of the images in each octave.

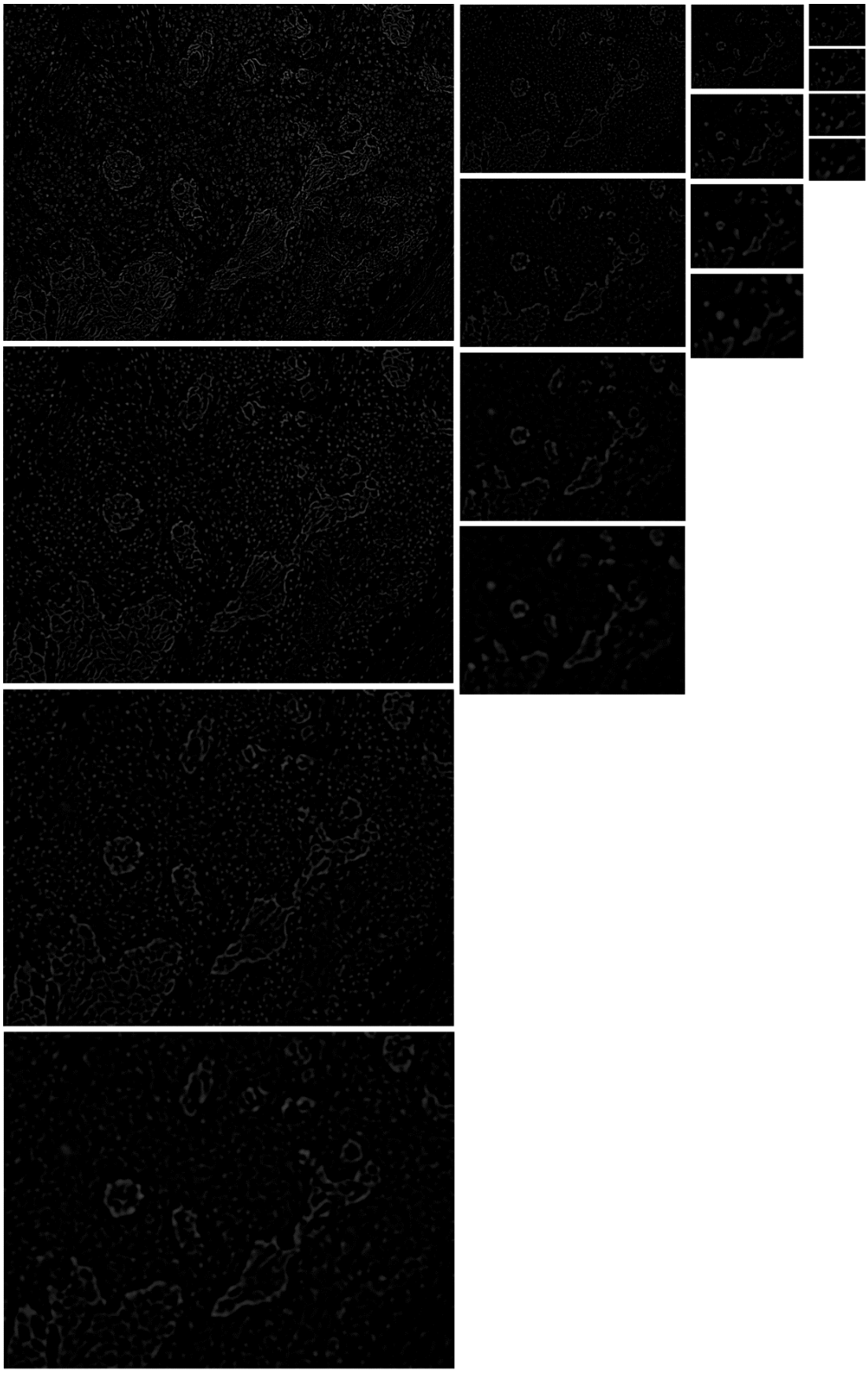


Figure 15 : DoG (Difference of Gaussian) pyramid. This pyramid has the same number of octaves as the Gaussian pyramid but each octave contains on less scale.

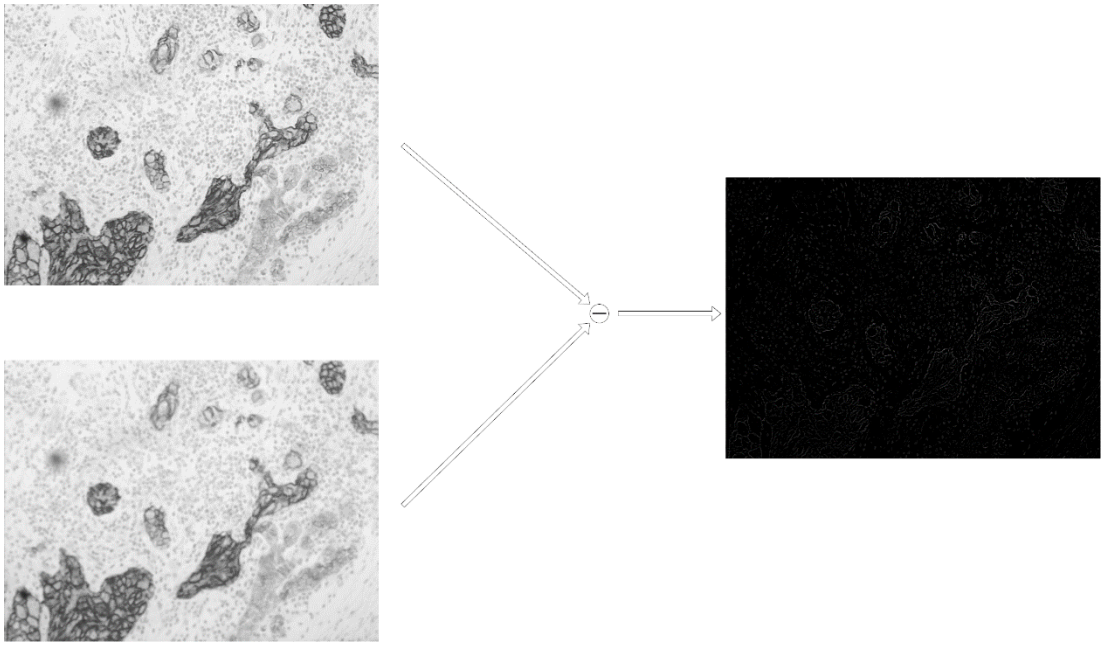


Figure 16 : By subtracting two neighbouring scales of the Gaussian pyramid, one scale of the DoG pyramid is generated.

The next task is the creation of the Difference of Gaussians (DoG) pyramid, which is produced by subtracting the neighboring scales in each octave, which is shown in Figure 16. It is possible to take advantage of some parallelism and compute the DoG pyramid alongside the Gaussian pyramid. The subtraction of the images will result in an image that will have all black pixels, except of the outlines that are going to have non-zero values, thus imitating the Laplacian filter. The final result will look like in Figure 15 where the pyramid once again consists of 4 octaves, each one having 4 scales.

Algorithm 1 Build Guassian Pyramids

Input: image s
Output: pointer to pyramid

initialize $k \leftarrow \sqrt{2}, \sigma \leftarrow 1.6$
initialize number of *octaves* and number of *scales*
for $i \leftarrow 0$ to *octaves* **do**
    for $j \leftarrow 1$ to *scales* **do**
        $\sigma' \leftarrow k * \sigma$
        $Gp(i,j) \leftarrow Blur(s, \sigma')$ ▷ Gp : Gaussian pyramid
        $Gp(i,j-1) \leftarrow Blur(s-1, \sigma)$
        $DoG(i,j) \leftarrow Gp(i,j) - Gp(i,j-1)$ ▷ DoG : Difference of Gaussians
    end for
    $Resize(s, 1/2)$ ▷ halve the image size
end for
return Gp, DoG

Now it possible to coarsely locate the maxima and minima points in the DoG pyramid. This can be done by comparing each pixel with its 8 neighbors that are on the same scale, but also comparing them with the 9 neighboring pixels in the scale above and the corresponding 9 the scale below as indicated in Figure 2, where x marks the pixel to be checked and the green circle are the 26 neighbouring pixels. Although an iteration through all the pixels is needed, in the majority of the cases, just from the first few checks, it will be sufficient to discard the non-maxima/minima points. By now, we have narrowed down the location of the keypoints, but still, they are not detailed enough. Among those, there are unstable keypoints that need to be discarded. First, to be cleared are the low contrast keypoints, and those are simply the keypoints whose pixel values are below a certain threshold we have defined, in this case $|D| \geq 0.03$, where $D(x, y, \sigma)$ is the value of a pixel in a certain layer in the pyramid. Next, keypoints to be filtered are those on the edges. The basic idea here is, using the gradients of the keypoints, there are three cases:

- The area surrounding the keypoint is flat. Because of that, the gradients (one on each axis x,y) will have low values.
- The keypoint is on an edge, which means that only one of the gradients will have a high value, the one that is perpendicular to the edge.
- The keypoint is on a corner and so it is expected for both gradients to have high values.

Based on that, it is easy to realize that the keypoints that are located on corners are more important and stable than others. Now using the Hessian matrix, it is possible

to compute the curvature ratio of the keypoint, and once again, if it does not satisfy a certain threshold (≥ 12.1), it will be eliminated. Having eliminated the more unstable keypoints in the image, the result should look like Figure 17. It should be noted that all those keypoints are not all from the same octave. This gives the ability to search for different kind of keypoints, since in the first octave we detect the finer details as opposed the next layers, which explains why there are some keypoints that do not seem to be located at a corner. Knowing in which octave is located each keypoint, achieves scale invariance. For example, if image "A" contains a cube but has double the size of image "B" which also has the same cube inside, the keypoints to be matched are not expected to be located on the first octave, but instead on one or maybe multiple octaves that follow.

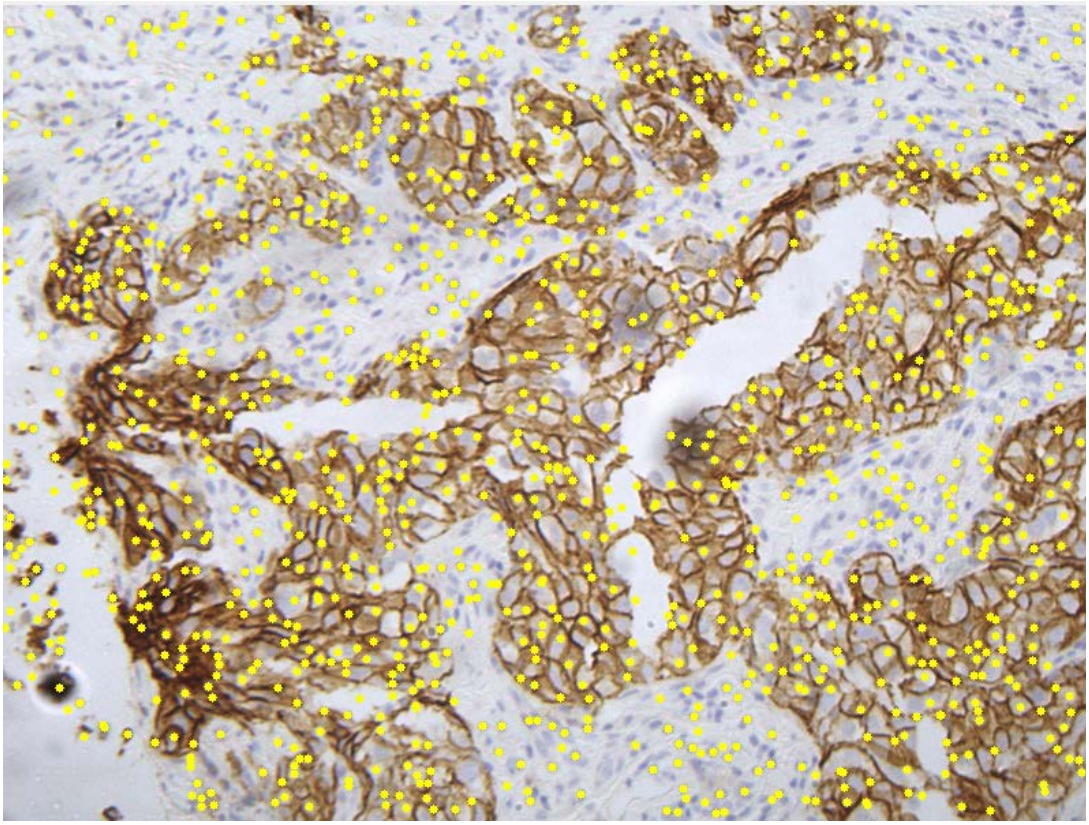


Figure 17 : Sample with 1179 keypoints.

Although due to the application of the microscope, images with different scales are not expected, this part of the algorithm could not be skipped and just use the basic octave, since apart from scale invariance, the pyramid provides keypoints of different

categories, like finer details in the first octaves vs. coarse details in the lower ones, that could be important to finding matching keypoints between images later.

Algorithm 2 Keypoint Detection

Input: Gaussian Pyramid (Gp) , number of octaves ($octaves$), number of scales ($scales$)
Output: list of keypoints

initialize $curvatureThreshold, contrastThreshold$
for $i \leftarrow 0$ to $octaves$ **do**
  for $k \leftarrow 1$ to $scales + 1$ **do**
    for each $pixel \in s$ **do**
      if $pixel$ is max/min in neighborhood **then**
        if $pixel \geq contrastThreshold$ **then**
          compute Trace of the Hessian at the extrema location
          compute Determinant of the Hessian
          $curvatureRatio \leftarrow \frac{Trace^2}{Det}$
          if $curvatureRation \leq curvatureThreshold$ **then**
            save keypoint
          end if
        end if
      end if
    end for
  end for
end for
return list of keypoints

The next thing to assign to each keypoint is an orientation. This process is important in order to achieve rotation invariance. The basic idea is to make the keypoints even more distinct by introducing one more parameter, apart from its coordination in the pyramid. By collecting gradient orientations and magnitudes around each keypoint it will be easier for the matching process in case two keypoints are relatively close to each other, there will be a possibility to distinguish the correct match from the orientation. First, the magnitude and the orientation of the pixels surrounding each keypoint, which will depend on the scale of the keypoint, are calculated and placed in bins of a histogram. The histogram consists of 36 bins that correspond to the 360 degrees of orientations and each bin will contain gradients from certain points. Every entry is weighted by the gradient magnitude. In other words, the orientation will give which bin a point will be allocated while the magnitude will inform about the significance of that particular point. A representative histogram of this kind is shown

in Figure 18, where the keypoint will have one primary orientation. After being accumulated, the orientation histogram is smoothed by applying six times a circular convolution with the three-tap box filter $\frac{[1,1,1]}{3}$ [14], and a parabolic interpolation will occur to make small corrections to the position of the peak. The interpolation will occur with the interest points being the two besides the main peak. Assuming that those three points are located at x1 = -1, x2 = 0 and x3 = 1 the procedure can be simplified into the following equations:

$$a = y2 - \frac{y1 + y3}{2},$$

$$b = \frac{y3 - y1}{4}$$

$$x_{peak} = \frac{b}{a}, \quad y_{peak} = y2 + \frac{b^2}{a}$$

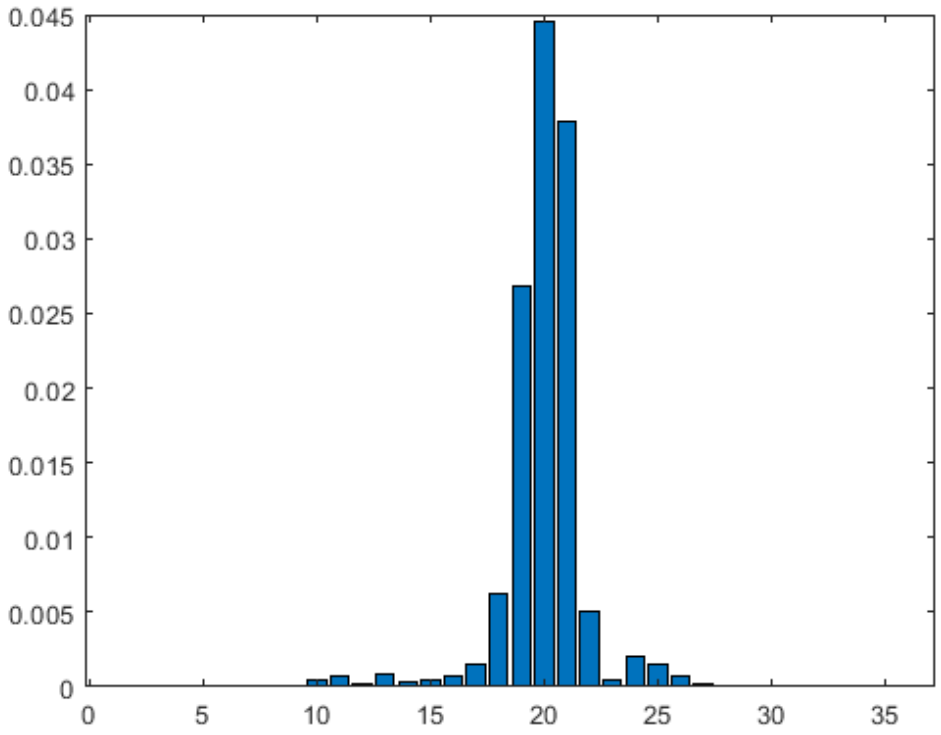


Figure 18 : Orientation histogram around a randomly selected keypoint.

Although cases such as in Figure 18 are the most common ones, there are exceptions where the histogram of a keypoint may have multiple peaks in the histogram and in case those picks are significant enough, another keypoint can be created at the same position but with different orientation, if the local peaks value is not adjacent to another peak (essentially checking if there is a peak) and if:

$$m_{local\ peak} = 0.8 * m_{global\ peak}$$

where m is the magnitude of the peak.

Algorithm 3 Assign Keypoint Orientation

```
initialize window length (w)
binCount ← 36
for each keypoint do
    for i ← keypoint.x − w to keypoint.x + w do
        for j ← keypoint.y − w to keypoint.y + w do
            gr_mag ← compute gradient magnitude
            compute gradient orientation
            id ← Find closest bin
            hist_orientation[id] ← hist_orientation[id] + gr_mag
        end for
    end for
    average histogram
    interpolate peak location
    Save keypoint orientation
    if ∃(secondary peak) then
        create new keypoint
    end if
end for
```

By now, all the keypoints have been located and assigned an orientation. The next to do is to create a unique "fingerprint" for each one. For this, we will take a 16x16 window surrounding the keypoint and divide it into smaller sections. Each section will be a 4x4 window where, once again, the gradients of the pixels inside will be computed and organized in smaller this time angle histograms. The new angle histograms will have 8 bins that range from 0-44, 45-89, 90-134, 135-179, 180-224, 225-269, and 270-

359 degrees. This way, we will obtain information not just for the keypoint itself but also for the surrounding area, thus further decreasing the probability of finding matching keypoints between two unrelated ones (false positive). The 16x16 window will also have to take into account and neutralize the angle of the keypoint, so cases where the same feature is located in two different images, but due to them having different orientations, no match between the keypoints is found (false negative) will not occur frequently. To achieve this, the first task is to also rotate the 16x16 window by the appropriate angle and locate the positions of the pixels in the image that will be inside the rotated window. For that information, the sine and cosine of the keypoint angle will be utilized and the formula is as follows:

$$x_{new} = x_{old} * \cos(k_{ori}) + y_{old} * \sin(k_{ori}) + k_x$$
$$y_{new} = x_{old} * (-\sin(k_{ori})) + y_{old} * \cos(k_{ori}) + k_y$$

Naturally, not all the 4x4 blocks around the keypoint will provide the same significance to the identification of the keypoint, since the further away a block is, the higher are the chance for the same block to be utilized again by a nearby keypoint, which can occur for multiple blocks at the same time. For that reason, a Gaussian weighting function can be used so the information given from the furthest gradients will have a more minor impact on the results. The whole procedure can be visualized in Figure 19 and Figure 20. In the first case (Figure 19), the blue lines of the grid define the borders of the 4x4 blocks aforementioned. In Figure 20 the representation of the grid is exaggerated in order to be easier to conceptualize.

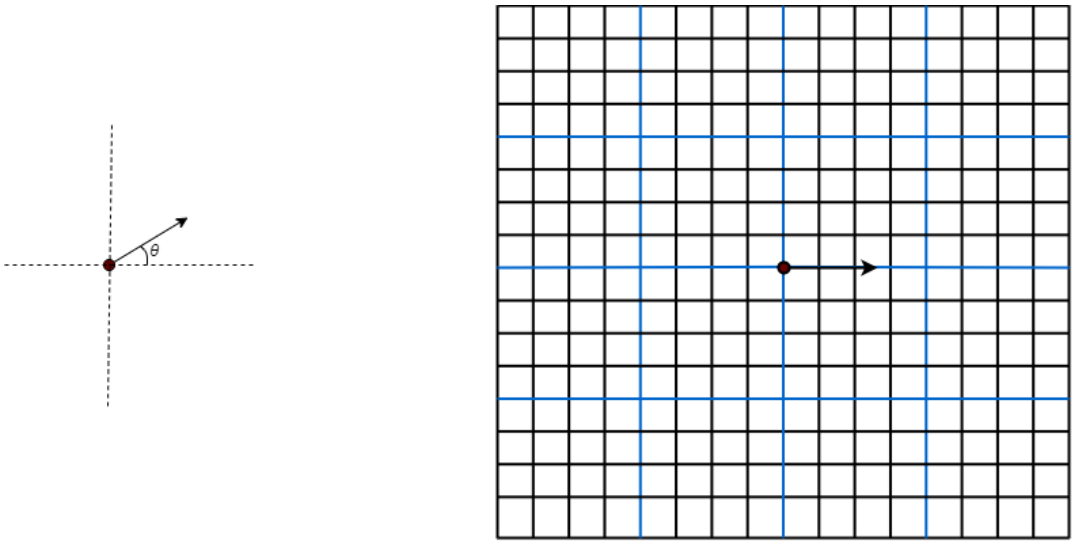


Figure 19 : On the left a keypoint is shown with its orientation. On the right the16x16 window around the keypoint is presented.

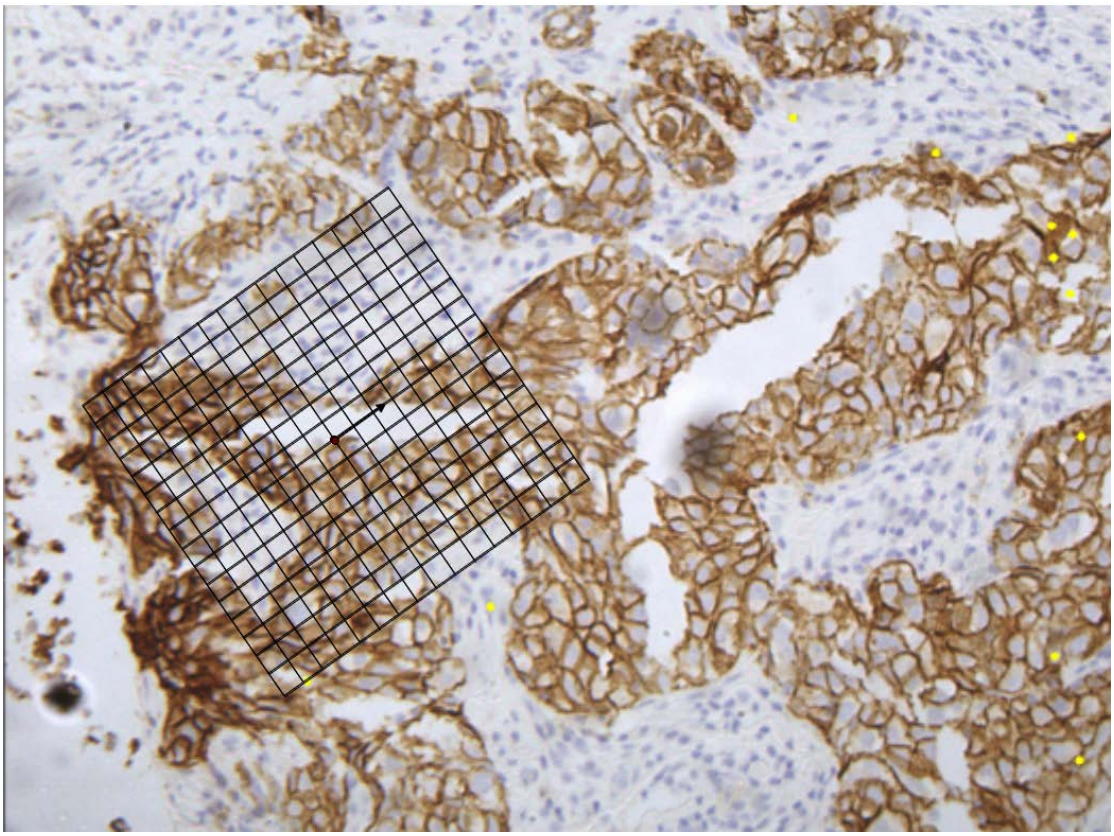


Figure 20 : This figure shows how the 16x16 window will look like after adapting to the keypoints rotation.

The last step to conclude the feature descriptor's creation, is to use a threshold of 0.2 as suggested by Lowe [4] and then normalize it.

Algorithm 4 Create Keypoint Descriptor

Input: Gaussian Pyramid (Gp), list of keypoints
Output: feature descriptor

```
for each keypoint do
    for each pixel in a window do
        comment:Compute new pixel locations
        x_new ← cos(k_ori) * x + sin(k_ori) * y + x_old
        y_new ← − sin(k_ori) * y + cos(k_ori) * x + y_old
        Interpolate new pixel location
        Apply weights depending on pixel location
    end for
    Normalize descriptor
end for
return descriptor
```

An important thing to have in mind is that aside from the functionality, the goal is to run the algorithm in a logical amount of time. If the characteristics of the microscope are recalled, the output images that it is able to capture, have high resolution, something that in terms of time needed to analyze each image, can be a hindrance. A countermeasure taken for this reason is the downscaling of the images that are inserted into the algorithm. This change will have an effect on the accuracy of the program since the smaller image in terms of resolution (less detailed) will probably mean a smaller amount of keypoints detected and thus lower chances of finding two matching keypoints between images. This of course, will also depend on the amount of downscaling that is going to occur, as well as depend highly on the contents of the image, and in the case of the microscope there are occurrences where the image is mostly empty, due to capturing the edges of the sample. Namely, the initial images that are captured have a resolution of 2500x1900 pixels, and the resolution after the rescaling can be either 1250x950 or 800x608. It should be noted that it is essential to keep the aspect ratio so that no distortion to the contents will occur due to the rescaling, especially since that kind of change can result in a doctor making a false diagnosis. After the analysis and even the blending process are done, the composite image can be rescaled again to the original resolution.

Another slight alteration that aims to the reduction of the execution time is to take advantage of the information given by the microscope and use them to introduce regions of interest (ROI) in the image. Since the microscope captures images of the

sample consequently and there is access to the movement's orientation, there no need to analyze the whole image every time. Instead, depending on the orientation of the movement, the proper region is selected from the image to be analyzed. Furthermore, when the microscope starts its automatic image capturing process, it will start from the left part of the sample and move on by capturing consecutive shots until it reaches the rightmost part of the sample on that particular line it started from. Subsequently, it will move down and to the left, making the reverse motion until it finds the end of the sample once again. This process will repeat until the whole sample is captured and a small part of it is showcased in Figure 21. In accordance with that knowledge, when performing stitching between images, it is possible to know which sides are overlapping, and thus limit the search for the keypoints as well as diminish the sizes of the pyramids.

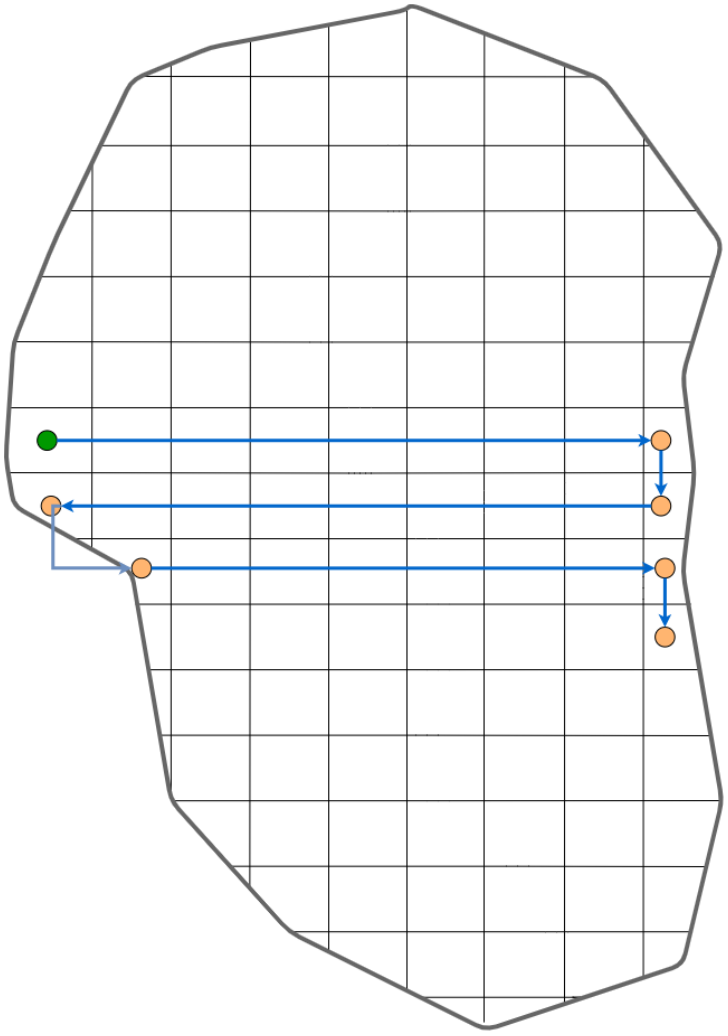


Figure 21 : Assuming this is a sample, the grid represents the pictures that will be captured by the microscope and the blue arrows indicate the path of the camera. The green circle is the starting point while the yellow circles are the points where the camera changes the direction of movement.

At this point, the implementation of the basic SIFT algorithm is complete. However, as shown in Figure 21, the current application may differ from the classical panoramic photography, where multiple pictures are stitched together and belong to the same line concerning the x-axis while having a slight deviation at the y-axis from image to image, producing the elongated field of view. In this case, depending on the sample size as well as the parameters set by the user (magnification), it rare for a sample to be able to fit in one row of images. In most cases, there will be multiple rows needed to cover the entire sample fully. This results in the need for vertical stitching. As for the SIFT algorithm, there is no need for parametrization to handle this case, as the algorithm operates without the need-to-know what mode is active (vertical/horizontal stitching), with the only prerequisite being to appropriately define the ROIs. The whole sequence of the stitching will change slightly. The camera will perform the same movement as before and the images will be stitched first in lines. Upon the end of capturing entire lines of images, the vertical stitching of those two outputs will be performed. The result of the vertical stitching is shown in Figure 22.

Another improvement that was developed this time for the vertical stitching, and it extends the application of the ROI. Generally, the vertical stitch is more time and resource-consuming than its horizontal counterpart and this can be explained by the different sizes of the ROIs. In the horizontal case, the images to be analyzed have approximately the same size regarding the image height (size of 1 image), whereas their width may vary (size of multiple images), but it is countered from the usage of ROIs that limit the width to be analyzed. In the vertical case, usually, the images to stitched are composites containing multiple images and thus having large overlapping regions, as shown in Figure 23. This extended overlapping region is not always needed to find a match between the images. Instead, a small part of it can be chosen to represent the whole line and checked for matches with the other composite image. In more detail, during the creation of each composite image (row of images), data are stored containing details of each individual image that is being merged, including size, position relative to the other images, etc. Using that information, it is possible to substitute the composite line image with one of its compounds and run the algorithm between the compound and another composite image as shown in Figure 24. From the list of images that are contained in a composite image, every image will be analyzed and compared until a match is found, this is done since the two composite images to be merged can differ substantially in length, so much that some images have no corresponding image in the following composite. This can be spotted in Figure 22. So, in the best case, the first image that will be checked will have a match with the next

composite image, and in the worst case, all the images will be checked from the list. This technique can substantially speed up the process (as long the worst-case scenario does not occur) since the row of images (composite) can become arbitrarily big in terms of width and even height. Since we limit the area to be analyzed, there will be a smaller number of keypoints and there will be a penalty to the accuracy. This technique could also be extended to both the composites, so in the end, only two compound images will be compared and the vertical stitching will have the approximately same results as the horizontal stitching in terms of time, but only in the best-case scenario where the first two images will find a match. Otherwise, up to n^2 comparison of images can occur to find a match. For this reason, the technique was used to only one of the composite images.

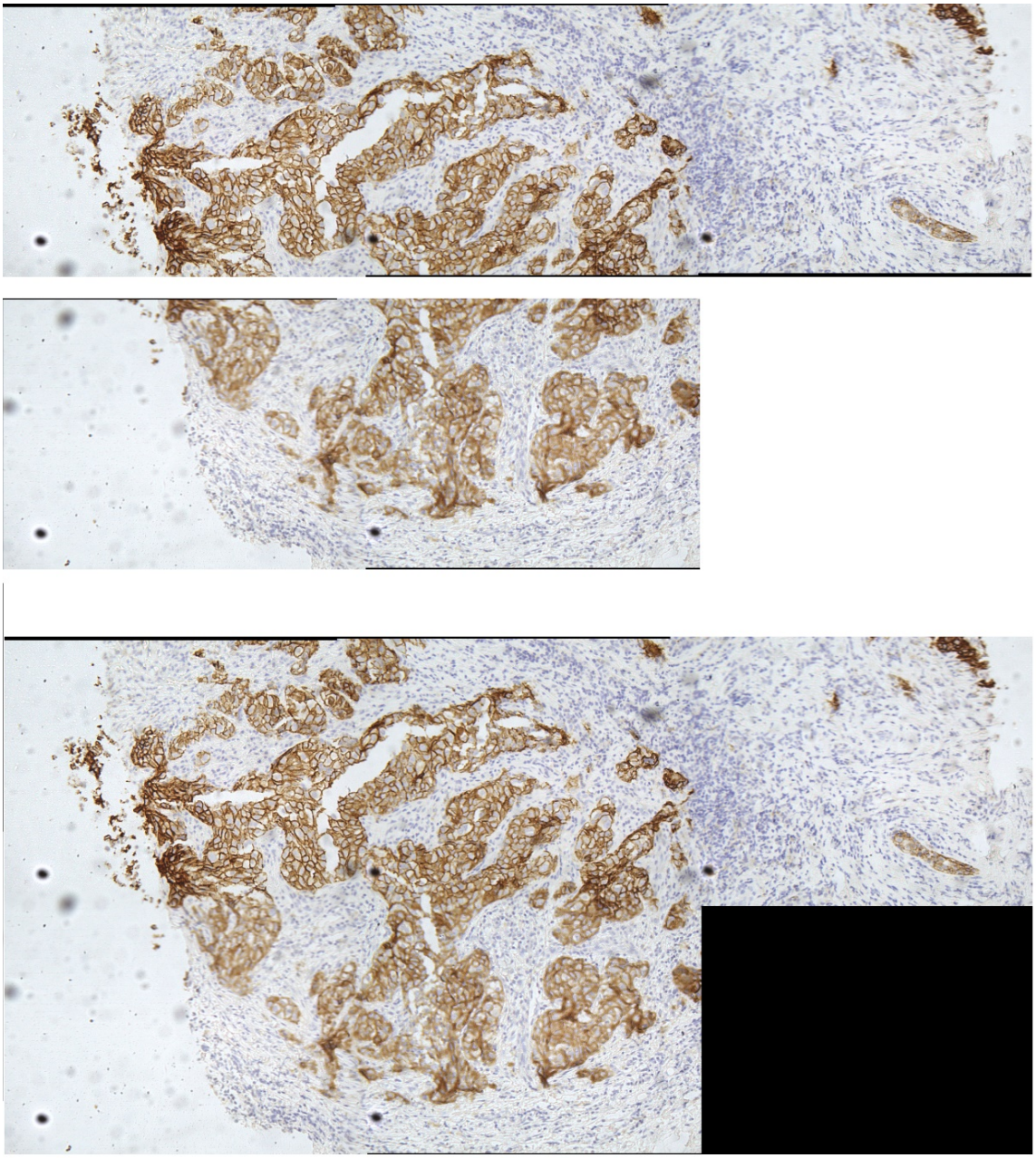


Figure 22 : An example of vertical stitching. The first image consists of 3 horizontally stitched images, while the second image contains 2 images. The third image is the result of the vertical stitch and the black box is created due to the width difference of the stitched lines.

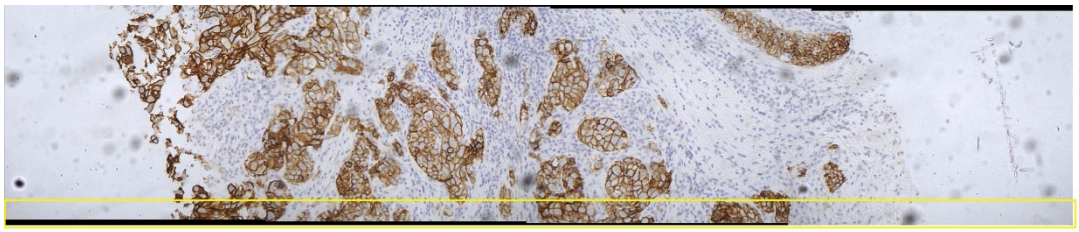
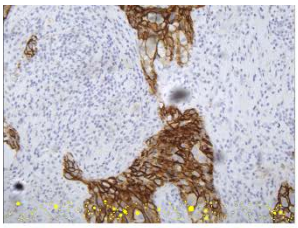


Figure 23 : This image is to be stitched vertically. The yellow designated area defines the overlapping region with the image to be stitched.



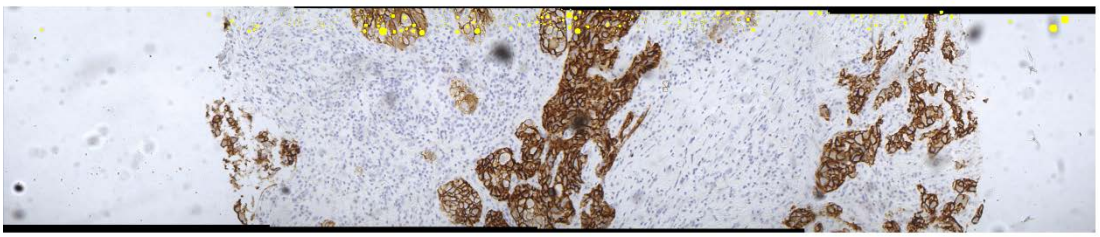


Figure 24 : Sample of vertical small stitch. The first image is part of a composite image (row of images) and is used instead. The second image is the second composite image that will be stitched with the one to who belongs the first image.

Algorithm 4 Small Stitch

Input: image S_1, image S_2, list of stitches l
Output: movement coordinates (dx,dy)

1: **for each** image $\in (l \cap S_1)$ **do** ▷ $l \cap S_1$: entries of the list that are contained in S_1
2: $SIFT(i, S_2)$
3: $Match(i, S_2)$
4: **if** Match successful **then**
5: Adjust Coordinates
6: **return** (dx,dy)
7: **end if**
8: **end for**

Another way to minimize the execution time of the algorithm the “blind stitching”. It was observed that when it comes to the horizontal stitching, the disposition needed to be correctly stitched is around the same with a deviation of 3 to 4 pixels. So, harnessing this knowledge, it is possible to skip the SIFT algorithm for some images when a stable distance is found. This can substantially speed up the stitching process but significantly increase the chance of erroneous results. For this reason, periodically, a standard stitching will occur (using SIFT) to those correct distances are used and rebalance if there is a need. When referring to distances, at this point, they are the x-distance and y-distance that the images need to be moved to be stitched. At first, only the first row of images captured were used to find those distances, and they would result from averaging the stitching results up to this point. This was fairly unstable since there were occurrences where the first line was too small to produce a good average distance. Another approach that was used was using the 10 first successful stitched results as a guide to finding the most frequent feature distance using a tolerance of 1 pixel on each side. Having that distance as a guide, the blind stitching process can follow. The rest of the distance matrix entries will not be discarded since every time a regular stitch (SIFT) occurs, the matrix will progressively rebalance itself by taking into account every new valid result. This means that even if at the start of the stitching process there are some consecutive erroneous stitch results after given some time, the correct distance will be used.

3.3.2 Matching (Stage 2)

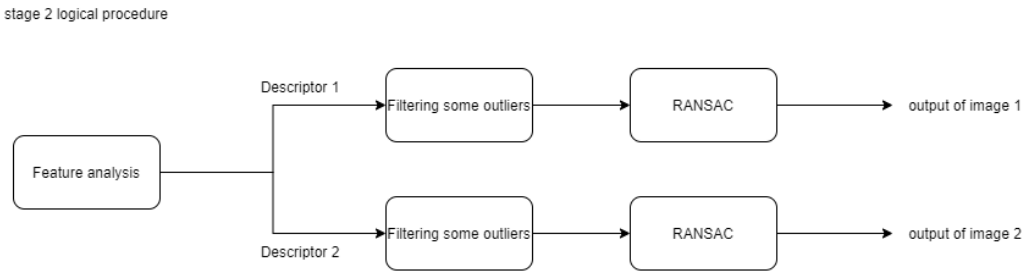


Figure 25 : A flowchart of the matching process. The output of the procedure will be the distance in pixels of the matching keypoints.

At this point, the feature analysis process implantation has concluded and the output of the previous procedure is one feature descriptor for each image that was inserted. However, until now, only the contents of each individual image have been examined, not the relationship between the two images inserted into the program. For this reason, the matching process was implemented and its output will have information about the distance of the matching keypoints of the 2 images if there are any, or in other words, the displacement that needs to occur in order for two matching keypoints to be on top of each other and merge.

Feature matching

The first step for the matching process is to utilize the information given from the process that took place before (feature analysis). For each keypoint in each image there is a matrix of 128 elements representing the descriptor of that keypoint. To identify which keypoints match between the images, it is needed to find which descriptors match. This is done by using the sum of squared differences (SSD) for the descriptors and using a threshold to decide if the descriptors are correlated. The threshold value was chosen through experimentation and the final value was $threshold_{ssd} = 100$. This will be a first filtering of the descriptors, but multiple may still satisfy this threshold and yet referring to different keypoints. So, the next step is to choose from the descriptions that remain, the one that will have the minimum SSD. However, there is still a case that a descriptor is not appearing in both images, so choosing the minimum SSD is still not sufficient. For that reason, another strict threshold is used with a value of one. Having found the matching keypoints, every pair will be inserted in a list containing information of the id of the matching keypoints and their SSD. The following analysis will distinguish the inliers and outliers of the matched pairs. The metric that will be used as the criterion that divides the sample points (pairset) will be the squared Euclidean distance. The most reliable method to find the inliers for this problem is to perform a full search, search every matched pair with each other and calculate the squared Euclidean distance and if the result satisfies a certain threshold that is chosen depending on the application, then we proceed to calculate the number of inliers and choose one of them to be representative. In this application, the threshold can be flexible using a value of 500 as a reference, since when a mismatch occurs typically the distance will have values of 40000. Although with this method the highest accuracy is achieved, the computational cost is quadratic ($O(n^2)$).

Algorithm 5 Full Search Matching

Input: descriptor d_1, descriptor d_2
Output: matching pair of keypoints

```
for each keypoint ∈ d1 do
    for each keypoint ∈ d2 do
        ssd ← Calculate ssd of d1, d2
        if ssd < threshold then
            matchedpair ← append pair of keypoints
        end if
    end for
end for
for i ← 0 to matchedpair.size do
    for j ← 0 to matchedpair.size do
        if i ≠ j then
            d ← Squared Euclidean Distance(matchedpair[i], matchedpair[j])
            if d < threshold then
                increment number of inliers of matchedpair[i]
            end if
        end if
    end for
end for
v ← find matchedpair with max inliers
return v
```

RANSAC implementation

An alternate way to distinguish the matched pairs of keypoints into inliers and outliers is the RANSAC algorithm. For this implementation, we will choose the minimum number of samples. In this case, the least number it needed is 2 matched pairsets. The randomly chosen matching pairs will be considered as inliers, and the square Euclidean distance will be the guide for choosing the rest of the inliers. Having the inliers chosen, we will cycle through every pairset that is not an inlier and compare it with one of the inliers. In case, the distance of the candidate inlier satisfies a threshold, then it is also added to the pool of inliers and naturally, if it exceeds the threshold value, it will be considered an outlier. This procedure will repeat itself choosing new inliers every time until a certain number of repetitions are completed, that is calculated from the RANSAC equation (2.3.1), and it depends on the probability of any sample being an outlier, the (outlier ratio) and the minimum number of samples and the desired accuracy of the algorithm. Assuming the outlier ratio of the images is 20%, then only 5 repetitions are needed to achieve 99% accuracy. Once the appropriate number of repetitions is done, the candidate that produces the highest number of inliers will be

considered correct. In Figure 26 a more graphical explanation of the RANSAC algorithm is represented. The red line indicates the pairset that is assumed as inliers, consequently every other pair will be compared with the red line and only those that match its orientation will be considered inliers (green lines). So, in the first case, the pairset will have 0 inliers, while in the second case 6 inliers are found, which indicates that the second choice is superior.

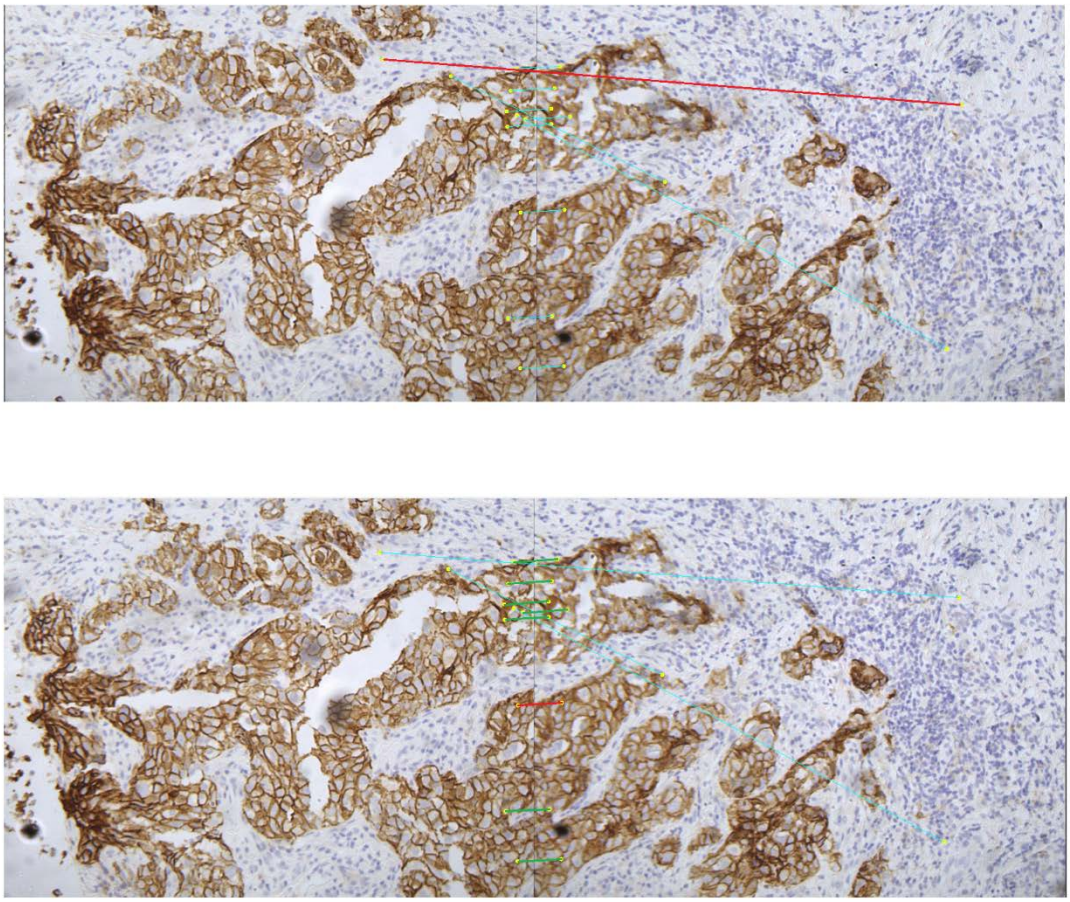


Figure 26 : Both images are different iterations of RANSAC. The matching pair of keypoints are represented with the blue lines. The red line is one of the chosen as inliers matching pair of keypoints, while with green line are highlighted the pairs that are inliers in accordance to the red line.

Algorithm 6 RANSAC

Input: descriptor d_1, descriptor d_2
Output: matching pair of keypoints

```
matchedpair ← create list of matching keypoints
n ← initialize number of iteration needed
for i ← 0 to n do
    Choose 2 random inliers (x_1, x_2)
    d ← Squared Euclidean Distance(x_1, x_2)
    for j ← 0 to matchedpair.size do
        if x_1 ≠ matchedpair[j] then
            d' ← Squared Euclidean Distance(x_1, matchedpair[j])
            if d' < d + threshold then
                increment number of inliers of x_1
            end if
        end if
    end for
end for
v ← find matchedpair with max inliers
return v
```

During the implementation of the Full Search method (FS), and due to a logical error, abnormally high values of the squared Euclidean distance were observed, even for inliers. This would result in the majority of inliers being rejected and have an image filled with outliers. In the first attempt to resolve this problem, the thresholds were adjusted and relaxed in order for the inliers to be accepted, although this resulted in increasing the chance of an outlier also being accepted as an inlier. This eventually led to cases where multiple samples had a large number of inliers and a lot of cases where multiple points had the same number of inliers, but different orientations, and in case the wrong “inlier” was chosen, then erroneous results were visible in the blending process. In an attempt to minimize this effect, a grading system was developing for the inliers, taking also into account the squared Euclidean distance when two or more “inliers” with different distances would exist. This way the sample with the maximum number of inliers and the minimum distance will be considered as the true inlier. The accuracy of the method was measured at 80%, which is suboptimal in this application. Although with this implementation, the initial results were somewhat promising, determining the true inliers, when the logical error was fixed, this method became obsolete and just provided additional computational load. In addition, even if the

accuracy could increase with better parametrization, the whole process was abandoned and so, it was omitted from the final application.

Case of Failure

Generally, the SIFT algorithm, as mentioned earlier, is described by its high accuracy and the ability to produce a higher number of keypoints, compared to other similar methods. Also, two different algorithms for matching have been reviewed up to this point, both having above 90% probability of finding a good match, assuming the parameters were chosen appropriately. Nevertheless, there are occurrences where the two images will be inserted into the program and no match will be found. Such cases are most probable to happen when the images that were analyzed are located on the edge of the sample that the microscope is reviewing, since the background of the sample is empty and no feature can be detected. The case that was just described does not occur only when the image is empty, but even if the image contains a portion of the sample and that portion is small enough, then once again, there is the possibility of failing. Since the usage of ROI has been adopted, if a stich is attempted with the wrong orientation (vertical instead of horizontal), once again the algorithm most likely will not find a match. Finally, even if the orientation is correct and the image contains a sufficient amount of sample, there is still a chance of not finding a match. When such a case occurs, depending on the level that the matching process failed, there is a possibility of correcting such an error. When the descriptors of the two images are compared and produce no pair of matching keypoints, then it was observed that by repeating the SIFT process can generate different results in the descriptors and consequently increase the matching pairs found between the images. This method will increase the computational time since the process will be repeated but only by the amount that SIFT needs to be completed, and it is used as a last resort allowing up to three repetitions for each pair of images.

3.3.3 Blending (Stage 3)

The output of the matching process will be the set of inliers, the best matching pair of keypoins of the two images. The basic idea of the blending process is to use that information and extract the disposition of the x-axis and y-axis that the images

need in order for the two keypoints to merge and be located at the same point (x,y). As mentioned in section 2.4, there are numerous algorithms designed for that purpose, but due to the nature of this scenario, the feathering method was chosen as the most suitable for the application. Since the images provided from the microscope are calibrated in a way that no rotation or different size of images are expected to be inserted, the application requirements are limited to location finding and solving the merging side-effects that may occur (seams, etc.). So, there is no real need to use a more sophisticated algorithm that results in an unnecessary increase of the computational time. Additionally, the mentioned algorithms were designed for more general cases of merging images, where the merging point can be anywhere on the image (even at the center), while in the scenario of the microscope, the merging will take place only on the edges of the images. Another argument that could be raised for choosing the feathering method is because it does not solve the problems of blurring and ghosting, which are not expected to occur in a successful stitch, it will be probably more recognizable by the user that the stitching made an error, rather than guiding the user to the wrong conclusion due to an alteration on the image of the sample, that occurred on the process of blending the two images.

The basic idea of the feathering algorithm is that the merged pixel intensities will result from a combination of the intensities of the individual pixels of the two images at the merging point after applying weights on each one. In this method, the closest a pixel is to the center of the image, the more significant its contents are, and thus, the closest to the edge each pixel is, the less value it has. So, in the overlapping region of the composite image, each pixel will be weighted relevant to its location in the compound image. This primarily solves the problem of having different brightness levels between the images and does not produce a seam when the merge occurs. For the implementation of the method, the first step is to choose one of the compound images as a reference. For example, when horizontal stitching is performed, then the leftmost image can be the reference. Any pixel that is not located in the overlapping region will take its value from the image it corresponds to by copying the contents of the compound image. While the pixels reside in the overlap region, their values will be calculated by the following equation:

$$i_{copmosite} = i_A \frac{(width_{ref} - x)}{|dx|} + i_B \frac{\left(x - (width_{ref} - dx)\right)}{|dx|}$$

Where, $i_A(x, y)$, $i_B(x, y)$ are the values of the pixels of image A and B respectively at (x, y), $width_{ref}$ is the width of the reference image and dx is the output of the matching method and the movement the images need to make in order to match.

The output of the matching algorithm will produce a set of dx and dy, that is the distance the images need to be shifted. Those can be positive, negative or even equal to zero depending on the type of stitching that is performed. In the case of the horizontal stitch, if the dx factor equals to zero, that would mean that there is no overlapping region between the images and they will be placed right next to each other, a negative value would mean that in the composite image, there will be a gap between the compound images (again no overlap region exists), while with a positive dx the images will overlap (typical outcome). The dy factor will translate into a difference in the position on the y-axis. The best-case scenario where $dy = 0$, there will only be the dx factor to consider, but in any other case, the two will not align perfectly and a set of pixels in the composite image will belong to neither of the two compound images, and since there is no information about the contents of those pixels it was decided to assign them with a value of zero (black regions). Depending on the sign of dy two cases of horizontal stitching can be distinguished as shown in Figure 27, and each case needs to be handled properly, so the correct region is assigned with the right values. The phenomenon of the black regions is generated by a slight offset of the microscope's camera angle relative to the stage where the sample is placed, which can be showcased in Figure 28. If the camera has even a very small angle on the z-axis compared to the stage, in the 2D representation of the image, a small dy offset will take place and will stay the same each time a new is performed, increasing each the black region by a small margin. So, the black region can through off the blending process by little every time, and grow increasingly larger, although it is easily corrected by adding the same offset to the dy factor. The same phenomenon will also appear in the vertical stitching process, this time the black region will form on the x-axis, but ultimately the solution will remain the same. However, this is not the only

way the black region will affect the vertical stitching, since some of them may appear from the horizontal process. When the vertical starts the ROIs that are defined will also need to be adjusted, otherwise there is the risk of excluding the overlapping region of the images from the ROI if the black region has a certain size. As to the final result, the black regions will appear only on the outer layer of images, that are primarily free of sample, will in areas filled with sample any black region will be covered by overlapping images.

The blending of the vertical stitch is slightly more complex than the horizontal case due to the side effect of performing the stitch using one reference image to represent a whole line of horizontal stitched images. When applying this method that was analyzed in section 3.3.1, the size of one of the images that will be stitched (the one represented by the reference) will be lost and also the location of the reference image. In other words, the results of the matching process will not be complete and further correction of the dx, dy will be needed. For this reason, when using the "vertical small stitch" method, a list is used to store the information that would be lost. For example, the first horizontal stitch produces a composite image s_1 from 5 individual images, while the second horizontal stitch results in image s_2 that is made from 6 individual images. When the "vertical small stitch" is performed and assuming that the s_1 is replaced by the 3rd in-line image then if a match is found with the s_2 then the resulting dx, dy will correspond to the start of the reference image (3rd), which is not the start of the s_1. By storing the blending points ($bp = image\ width - overlapping\ region$) and not just the sizes of the images already stitched, the location of the image will be available to use in the blending process.

Algorithm 7 Blending

Input: Movement coordinates (d_x, d_y), Stitch orientation s_{ori}, image s_1, image s_2

Output: Stitched image (f)

$f.size \leftarrow s_1.size + s_2.size - overlap$
for each $p \in f$ **do** ▷ p : pixel
    case $p \in s_1$
        $p \leftarrow s_1.pixel$ ▷ copy pixel value from s_1
    case $p \in s_2$
        $p \leftarrow s_2.pixel$
    case $p \in$ black region
        $p \leftarrow 0$ ▷ paint the pixel black
    case $p \in$ overlap region & horizontal stitch
        $p \leftarrow s_1.pixel * \frac{limit_1 - x}{|d_x|} + s_2.pixel * \frac{x - |limit_1 - d_x|}{|d_x|}$
    case $p \in$ overlap region & vertical stitch
        $p \leftarrow s_1.pixel * \frac{limit_1 - y}{|d_y|} + s_2.pixel * \frac{y - |limit_1 - d_y|}{|d_y|}$
end for
return f
comment: $limit_1$ refers to the width of image s_1 in case of horizontal, while height in case of vertical. x and y refer to the pixel location in image f

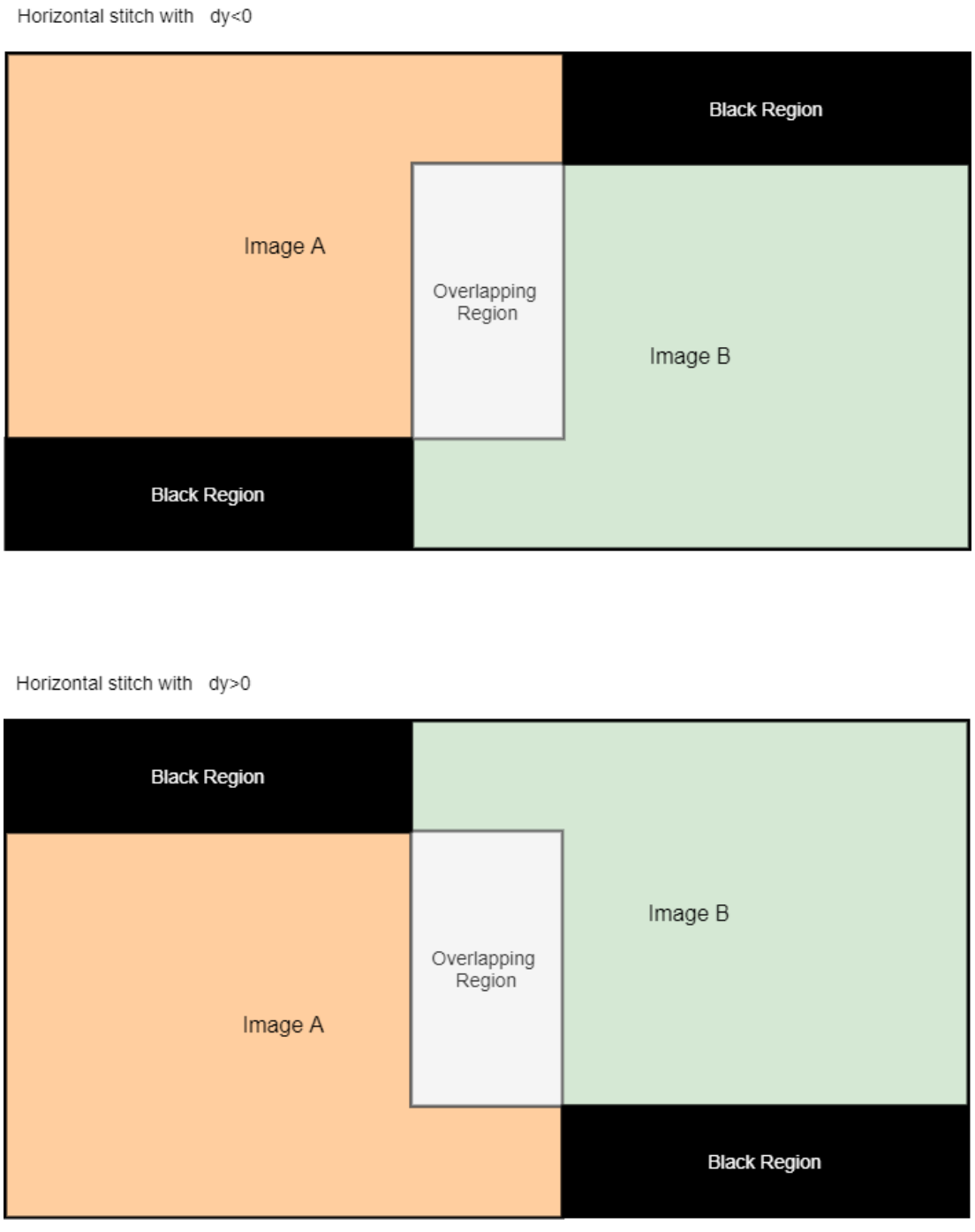


Figure 27 : Two composite images are shown and their contents divided into regions depending on the sign of the dy factor.

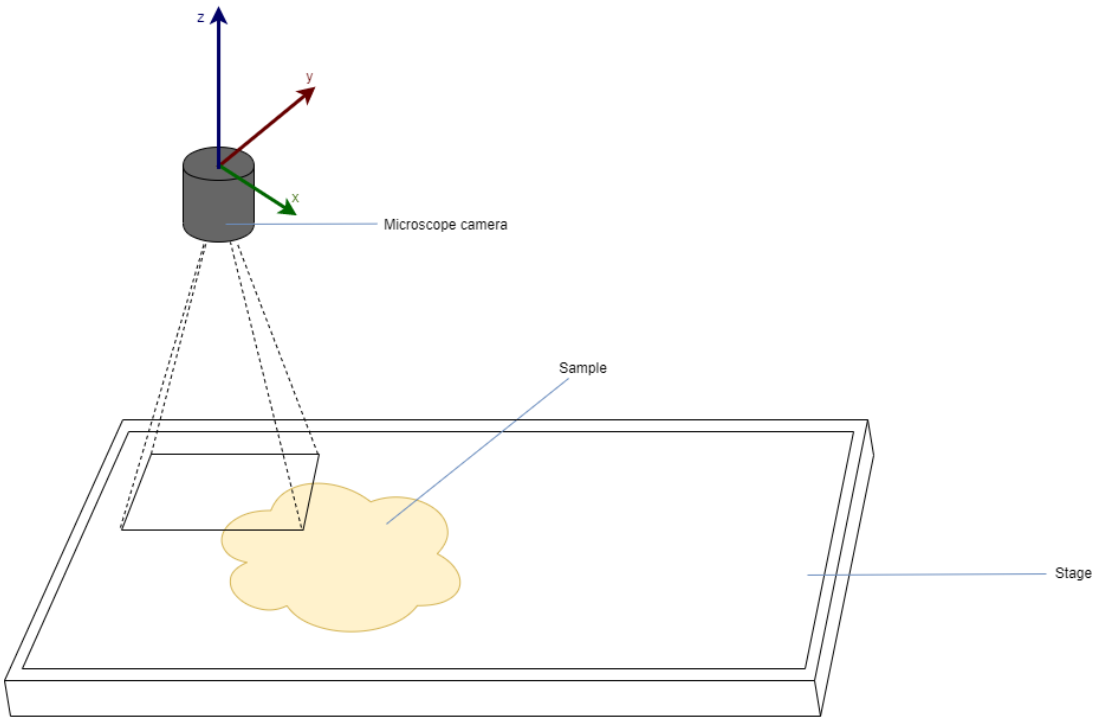


Figure 28: The position of the camera relative to the stage of the microscope.

4 Results

In the previous chapter, a review was done of all the algorithms that were used in this application. They were analyzed over their implementation and the expected results those methods should produce, and suggesting various improvements that mainly aim at the performance aspect. In the current chapter, the results of these algorithms will be investigated as well as the optimal parameter values that will yield the best results. Once again, the whole process can be divided into three major parts (feature analysis, matching and blending). The testing proceeded by taking a random sample from the microscope of 45 images and performing stitch on that set repeatedly while changing the appropriate parameter values. Of this sample, 35 horizontal stitches are to be performed and the rest will be vertical. The chosen sample contains a variety of cases, from images that have low feature density (located on the outskirts of the sample), and images with numerous features. The structure of the chapter consists of three sections. In the first, the results presented are due to parametrizing some values of the SIFT algorithm to perform more accurately to the scenario given. The second section takes place the parametrization of the matching algorithms and the view of their results. While the final section analyses the time of each process and scenario.

4.1 SIFT PARAMETRIZATION & RESULTS

The SIFT algorithm has numerous variables that need to be tuned in order to produce good results. Most of those parameters have been examined and their values determined by D.Lowe in his research [4], through testing in artificially created images in order to pinpoint the most optimal values, so the algorithm produces accurate and stable results regardless of the set of images that will be inserted as input. Some of those parameters include the number of octaves to be used along with the number of scales in each octave and both of them are important to achieve scale invariance. In addition, the amount of blurring (sigma) between each scale and each octave is also determined the same way. All of these parameters that were defined by the author of SIFT were not examined further in order to confirm their effect. Instead, the contrast threshold and the curvature threshold will be examined in order to better fit this particular scenario.

First of all, it is needed to determine the parameters that will help us decide whether a set of values produce a “good” result. The first thing that will help with that decision is the number of keypoints. However, this alone can lead to false conclusions. For example, if the thresholds to be used are too strict, many potential keypoints will be rejected, which could be crucial at a later stage, finding a match between the images. On the other, in case the thresholds are more relaxed, there is a possibility of keypoints being located that will lead to overshadowing the important ones and producing a false result during the matching (false positive). For this reason, along with the number of keypoints, also the accuracy (success rate) will be taken into account, even if that includes more using the matching process, and thus the test will not be entirely independent of each process. In Figure 30, the correlation of the keypoints and the contrast threshold is depicted, the latter of which is ranging from 0.02 to 0.8. As shown in Figure 29 and Figure 30, the smaller the keypoints number is, the probability of a successful stitch diminishes. A plateau is forming in the graph the accuracy that is indicating the region with the best results before the success rate starts dropping once again, this time due to a large number of keypoints. In Figure 31 and Figure 32 we have a graph that connects the number of keypoints to the curvature threshold and the success rate to the curvature threshold, respectively. Once again, there is a connection between the amount of keypoints discovered in the image and the success rate of the stitch. Too few keypoints, lead to errors due to not finding a match, while a large number of keypoints can lead to false-positive results. For the curvature threshold, the value that seems to provide with the best results is 10.

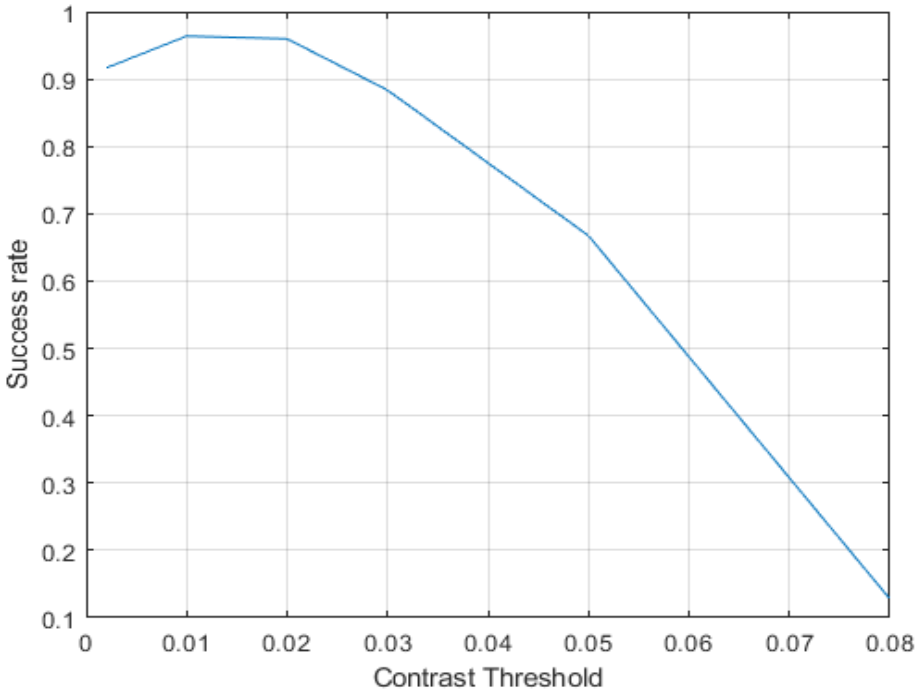


Figure 29 : The percentage of successful stitches depending on the contrast

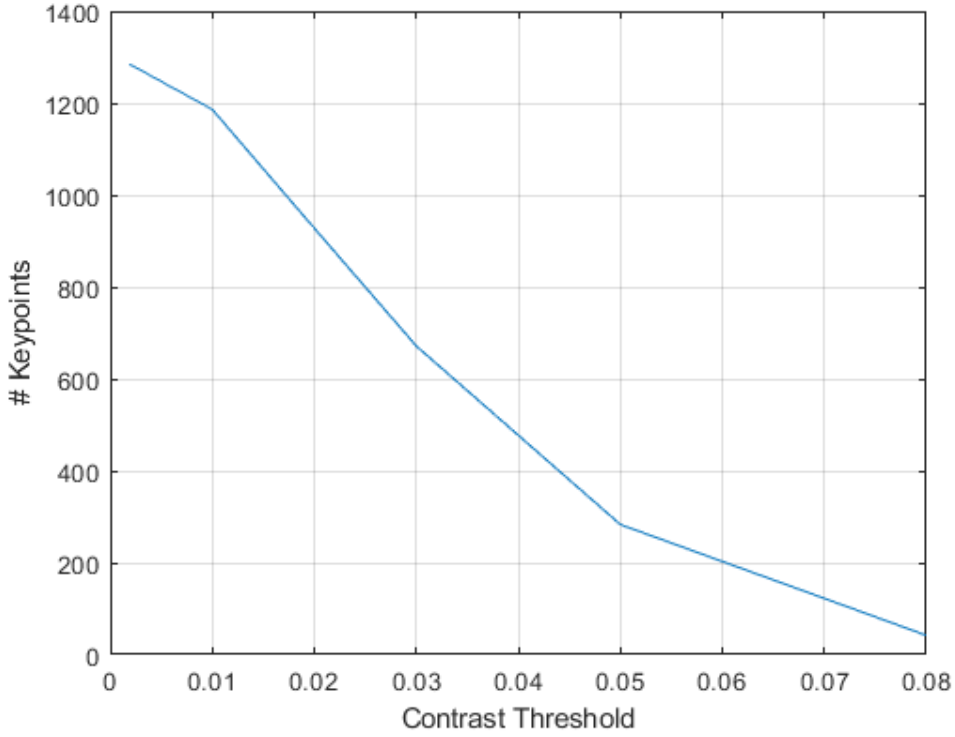


Figure 30 : Average number of keypoints, in horizontal stitch, relative to the contrast threshold.

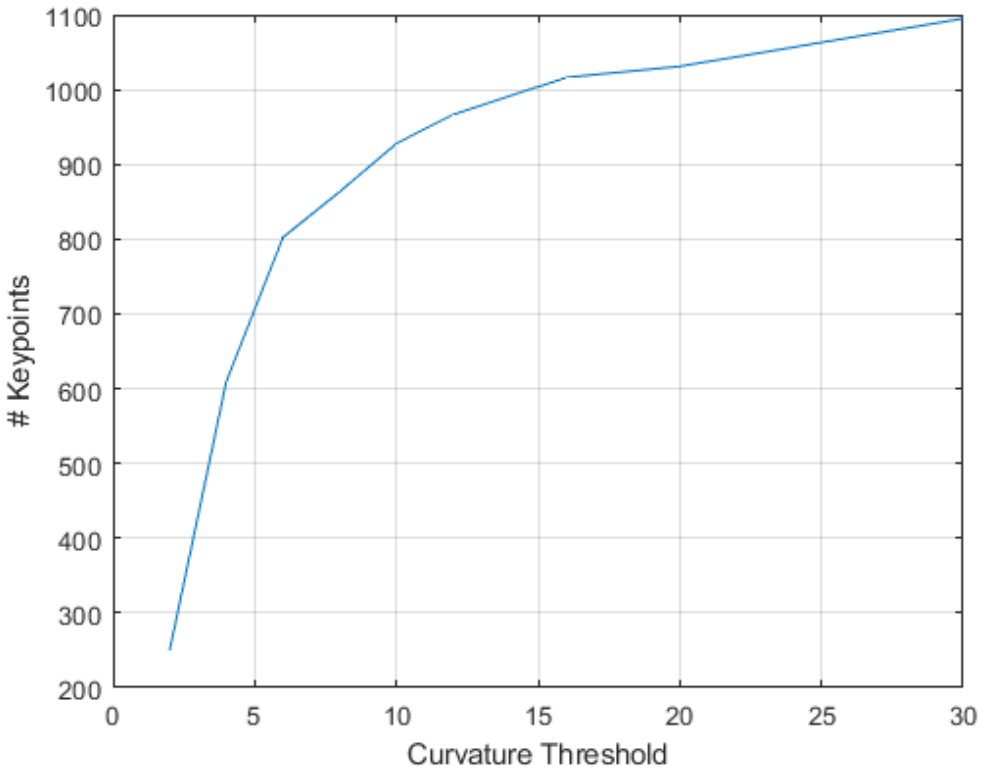


Figure 31: Average number of keypoints, in horizontal stitch, relative to the contrast threshold.

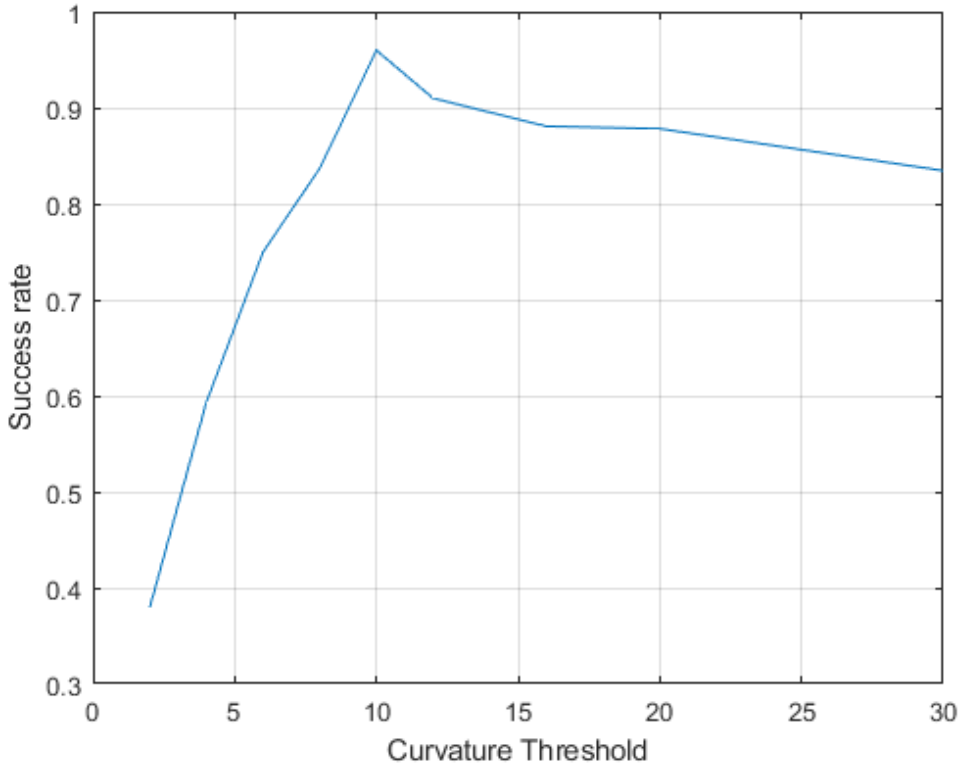


Figure 32 : The percentage of successful stitches depending on the contrast threshold

It should be noted that all four of the figures above (Figure 29, Figure 30, Figure 31, Figure 32) show the results of only the horizontal stitching. The reason that the sample is limited that way is because of the ability to ensure that all the images that contribute to those results have the same size. So, the only factors that will have an impact are the threshold variables that are under investigation and the contents of the images. The number of keypoints that are located during a vertical stitch will depend on the size of the images, along with the type of vertical method that is applied and will be multiple of the number detected at a horizontal version.

4.2 MATCHING PARAMETRIZATION & RESULTS

Similar to the previous paragraph, where some of the parameters of SIFT are tested to establish the best possible results, the matching process and all of the methods used contain numerous parameters that need to be tweaked. The way the results are produced is similar, as the same sample is used and the whole procedure is repeated multiple times in order find the average values that will unveil the most optimal combination of values. In this case, the two parameters that will be examined are going to be the matching pair size and the inliers produced by the different algorithms, and at a later stage, the elapsed time of each method will be analyzed. The matching pair size is created by comparing every keypoint in both of the input images and checking them for similarities. If the criteria are met, the pair will be a candidate inlier that will be decided later. So, the number of matching pairs can range from 0, and can even surpass the number of keypoints in the image, which is the case when a feature has multiple occurrences in the image. However, such a scenario is a rare occurrence. In most cases the pair size will not even exceed the keypoints number. The parameter that greatly influences the matching pair size is the feature descriptor gap threshold that is set. When creating a matching pair, the algorithm iterates through the keypoints in order to find the one with the minimum SSD (sum of the squared differences). If the final minimum SSD exceeds the chosen threshold, then the pair is rejected. The four

figures show the results of the method when the threshold value is fluctuating. Figure 33 shows the rate at which the size of the matching pairs of keypoints can be located. Interestingly, in the range 1 to 2.5, the rate is linear, while after exceeding the threshold of 5, the size that can be located between two images reaches an upper limit. The most important graph is the one depicted in Figure 34, where it is clear that for values smaller than 1.5 of the feature descriptor thresholds, the accuracy of the stitch is lowering rapidly due to the insufficient amount of keypoint matches that were established. On the other side, for values greater than 1.5, the success rate of the algorithm once again decreases as the threshold increases but with a smoother slope. Lastly, in Figure 35, it is shown how many of the mistakes depicted in the success rate figure are false-positive results, and in conjunction with Figure 34 the other two figures, when the size of the matching pair increases above a certain value, the list of keypoints contains a large number of outliers that if not appropriately recognized can lead bad stitching results. It should be mentioned that thanks to features provided by the microscope, it is possible to locate a false positive outcome even if the algorithm regards the results as successful stitching, due to the sensors that can give the accurate locations of each picture taken.

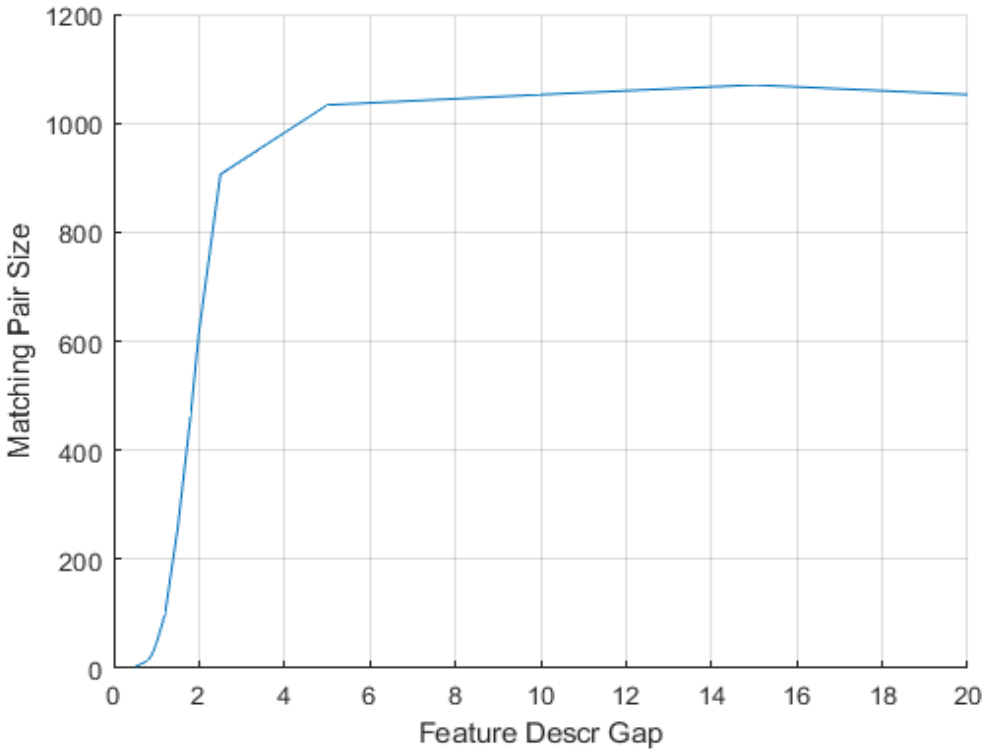


Figure 33 : This figure shows the correlation of the feature descriptor gap threshold with the size of the matching pair of keypoints.

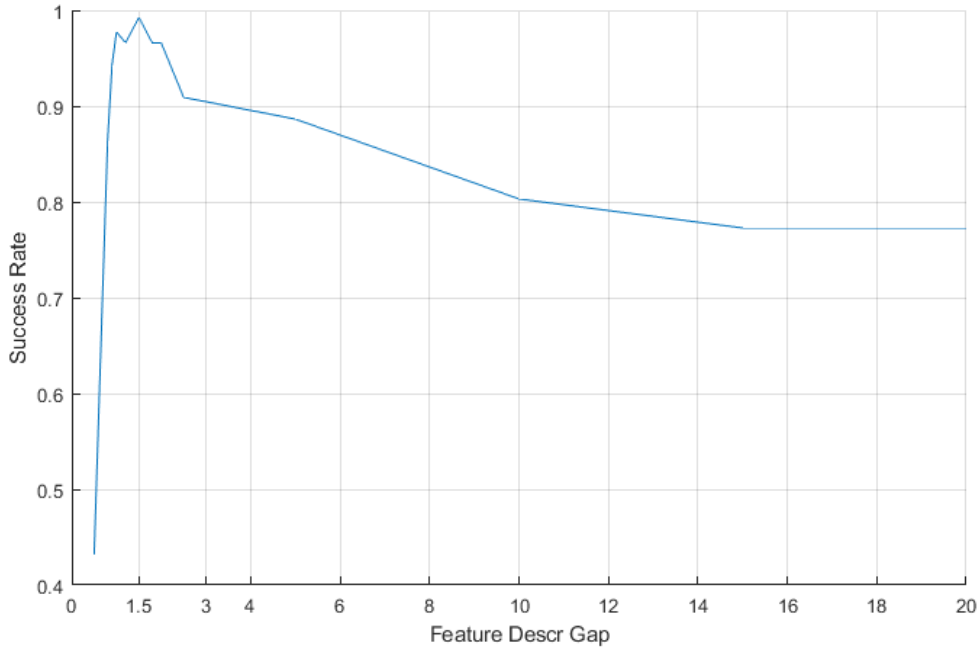


Figure 34 : The threshold can influence the accuracy of the stitch algorithm.

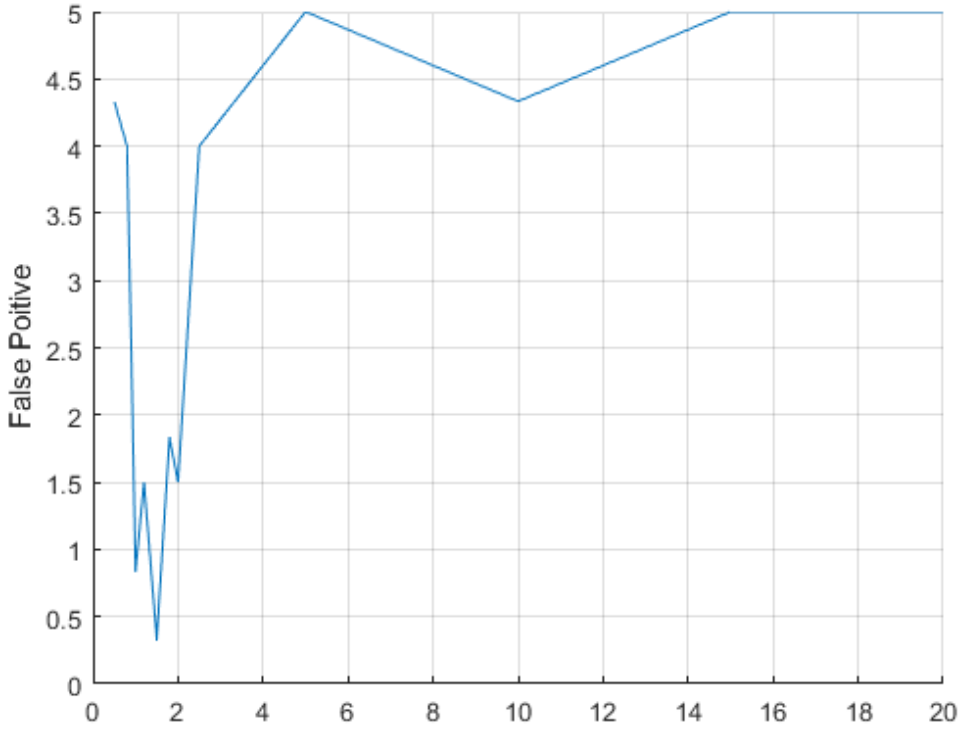


Figure 35 : This figure shows the amount of false positive stitching that were done during the test, while the threshold is changing. The false positives are included in the graph of the success rate.

Having finished with the parametrization of the feature gap thresholds that is responsible for the size of keypoint pairs, it is time to move on to uncover the most optimal threshold value that will discriminate the inliers from the outliers. The testing process took place using the FS method instead of the RANSAC that was also implemented. The reason for such a choice can be explained through the requirements of the test itself. RANSAC is described by its relatively high accuracy and the speed that is needed to find the inliers. However, in this testing process, where the intent is to check and find the optimal value for a certain threshold, the completion time of the algorithm is irrelevant and only accuracy is required. So, since RANSAC has a 99% accuracy, depending on the parameters chosen, there is the probability, although very small, for the algorithm to be unable to locate the correct set of inliers, thus making it not ideal for this scenario. An important thing to be mentioned at this point, is that in the test that follows, one more sample was used in addition to the one used in the previous test in order to provide additional information and help clarify some results. Starting with Figure 36, where the change in the number of inliers is shown. Although there may seem like significant fluctuations, the number is in a range between 17 to 22 inliers each time. This phenomenon can be explained through Figure 39 and Figure 40. The four different lines belong to the 2 samples that were tested, once using the “big” stitch method and once again using the “small” stitch method. Due to the fact that only a small portion of the image is overlapping, in the “big” stitch, many keypoints will be located that will not provide with helpfull information and the majority will be filtered by the Full Search or the RANSAC methods. In Figure 37, the plot shows how the accuracy is changing by altering the threshold, while in Figure 38 the number of false-positive matches is shown, and this time they are the only source of mismatch. By the guidance of figures 37 and 38, the value for the inlier threshold that seems to be more effective is under 500. Figure 40 is especially useful for the implementation of the RANSAC algorithm since, during the initialization there is need to determine the inlier ratio, and according to this plot, there are significant changes from method to method that will be taken into consideration during the time analysis.

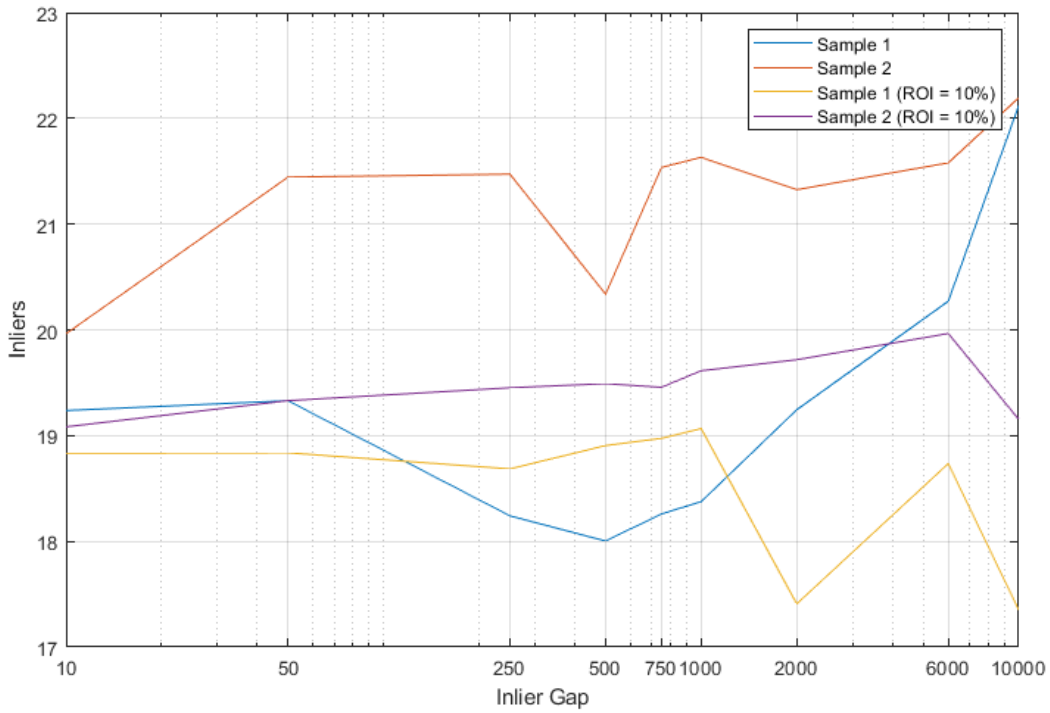


Figure 36 : Depiction of the fluctuation of the number of inliers located depending on the threshold.

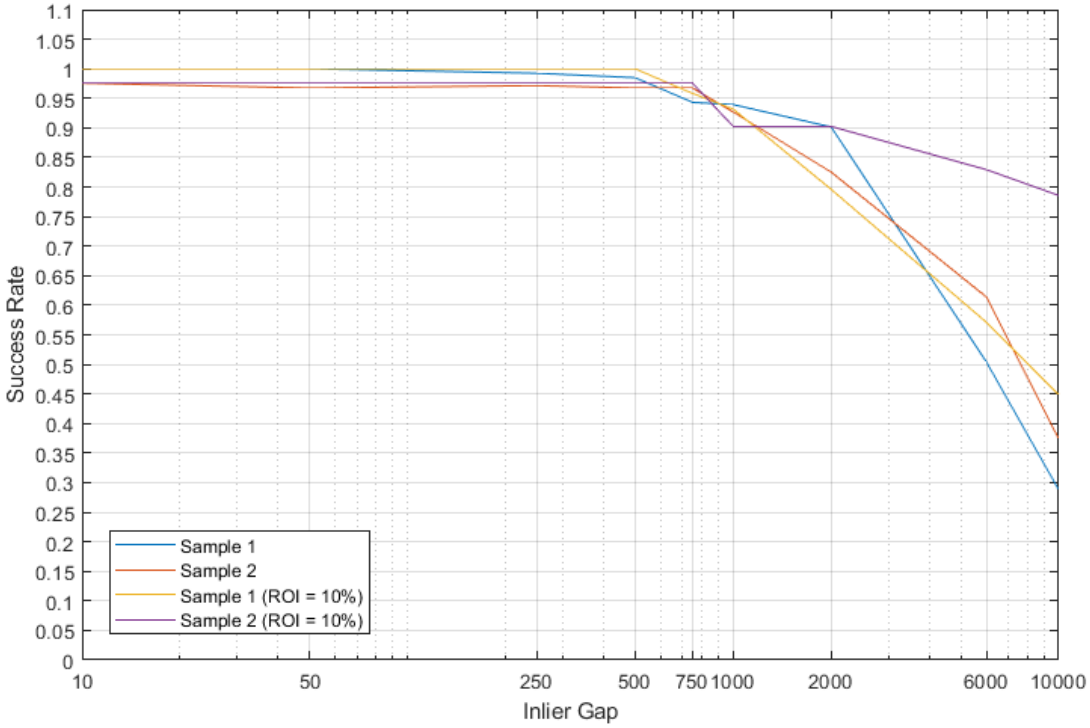


Figure 37 : The accuracy decreases as the inlier threshold increases.

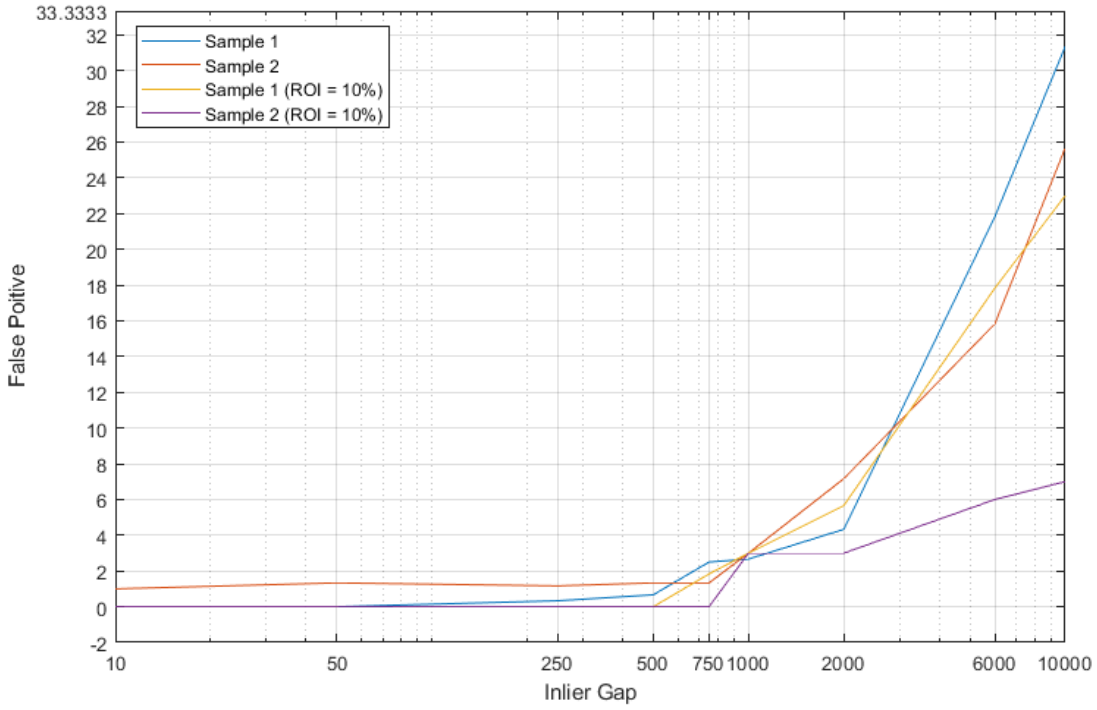


Figure 38 : This figure shows the number of mistakes created due to false positive matching.

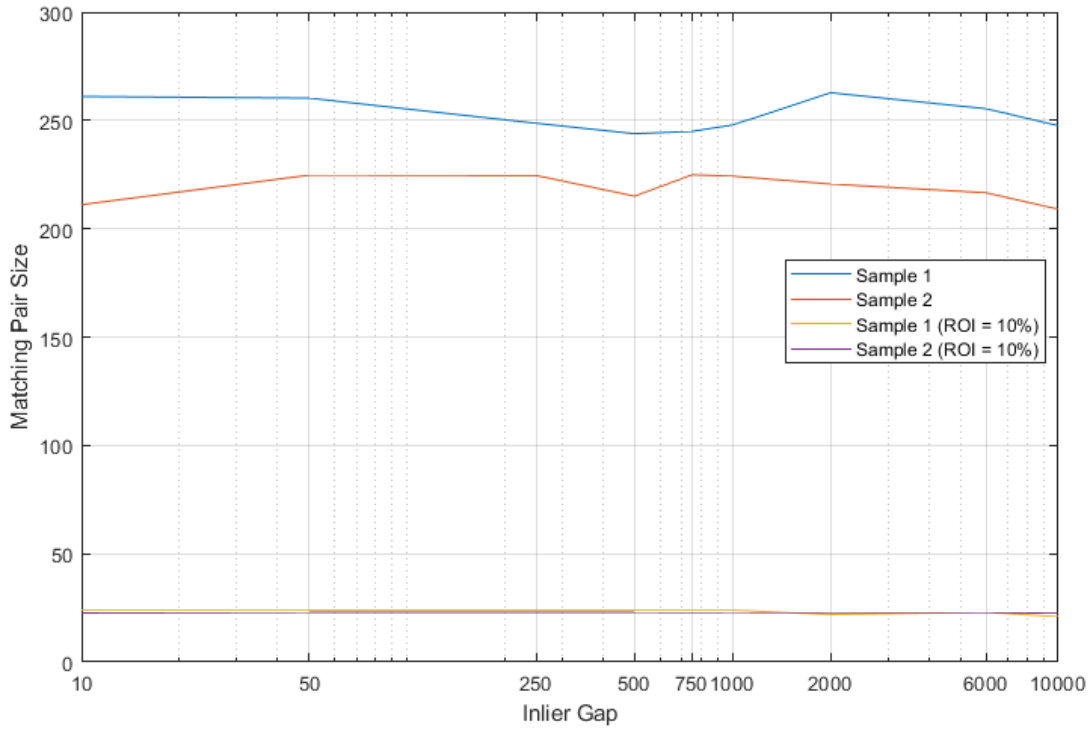


Figure 39 : This shows the difference of matching pair sizes between the “big” stitch method and the “small” stitch.

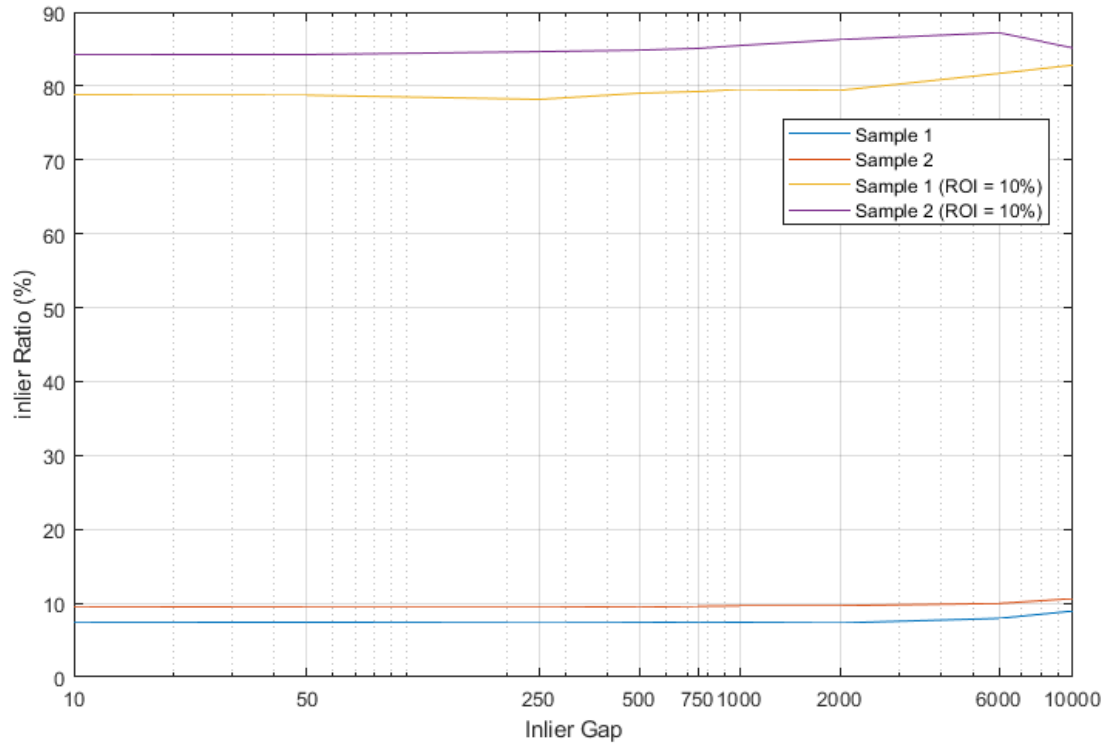


Figure 40 : The information given by this figure are important for the initialization of the RANSAC algorithm.

4.3 TIME ANALYSIS

Having finished with the testing process for finding the values that will provide the most promising results, we are able to move on to analyzing the time needed for each method to be completed. The same sample from the previous testing will be used, which contains 45 different images, and thus the stitching process will be called 45 times, 35 of which are going to be performed as horizontal, while the remaining 10 are going to be vertical stitches. For measuring the elapsed time of each process, built-in functions of OpenCV were used. The first one is the getTickCount(), which returns the number of clock-cycles after a reference event and so it is possible to calculate the difference between two reference points, the start and the end of each algorithm. After that, utilizing the function getTickFrequency(), which returns the frequency of clock-

cycles, or the number of clock-cycles per second. So, to find the time of execution in seconds, we only need to divide the difference of the clock cycles by the frequency. At this point the test will contain four different variations of the program, to uncover which of the techniques analyzed previously will provide with the best results. The four variations include a combination of “big/small” stitch and changing the region of interest each time between 100%, which covers the entire image, and 10% that includes the region of the images that overlap. Moreover, the tests will include different image sizes in order to measure the potential performance gain when using smaller images. Figure 41 presents the average time a stitch call will take depending on the type (horizontal/vertical). As expected, the lower the resolution of the input images is, the time needed for completion decreases, to the point of achieving up to 8 times faster completion time for the horizontal. The reason behind such a significant difference between the vertical and the horizontal timings can be explained by the structure of the vertical images. In most cases, the images that consist as the input of a vertical stitch will comprise of multiple images stitched together horizontally, thus having one image to be analyzed with a bigger scale, reaching up to ten times larger than the basic input image. According to that explanation it is easy to understand, that in case of a bigger sample (>45 images), the times for the horizontal will stay about the same, while the vertical timings are going to increase. Generally, utilizing the “small” stitch method seems to be more effective than narrowing down the region to be analyzed by the algorithm. Nevertheless, combining those two methods provide with the minimum elapsed time. In Figure 42, we see a repetition of the previous pattern, for the results, with the exception of the combination of methods “big” stitch with 10% overlap and “small” stitch with 100% overlap, where it is unclear which of those provide with the better performance. Especially in Figure 43, utilizing that small stitch method in combination with setting to analyze the whole image seems to not perform as effectively as other methods. This can be explained through the size of the area that needs to be analyzed in each method since a larger area will probably produce more keypoints, and thus more time is needed to check the compatibility of them and match them. When performing vertical stitch in this test, the average width of the images involved is about 4 times bigger in size compared to the basic input image of a horizontal stitch. In this case, the region in the image where keypoints can be found can be calculated and for the “big” stitch with 10% overlap, it is equivalent to analyzing an image of size 1000x760 (in case the original image is of size 2500x1900),

while in case of “small” stitch with 100% overlap, the image to be analyzed has a size of 2500x1900. Figure 44, Figure 45 follow the same principles mentioned before, for Figure 42, Figure 43. The big difference is that the runtime is limited to milliseconds. The reason of this similar pattern is due to the fact that, both full search and RANSAC algorithms are a direct extension of the matching process and they are highly dependent on the outcome that was produced by the latter. In Table 1, we compare the two methods for finding the inliers and all the contents of the table are in milliseconds. Overall, the RANSAC algorithm performs better, having a faster completion time that can be up to 10 times faster in some cases. That is a logical result since the RANSAC does not need to search the whole list of keypoints in order to locate the inliers. It is important to mention that during the tests, the number of iterations of the RANSAC algorithm were needed to change according to the method that was used since the different methods did not generate the same number of keypoints during the first match, and consequently, the inlier ratio changed. Lastly, Figure 46 shows the time needed for a blend of two images to take place. Contrary to the previous figurers, the current one does not show any particularly important differences between the performance of each method used. The reason behind this is that the methods are intended to reduce the runtime of the SIFT and matching sequences, so they do not affect the blending process. The only difference that takes place in this figure is the change in the size of the images, where the bigger the image to be stitched, the longer will be the time needed to complete the blend.

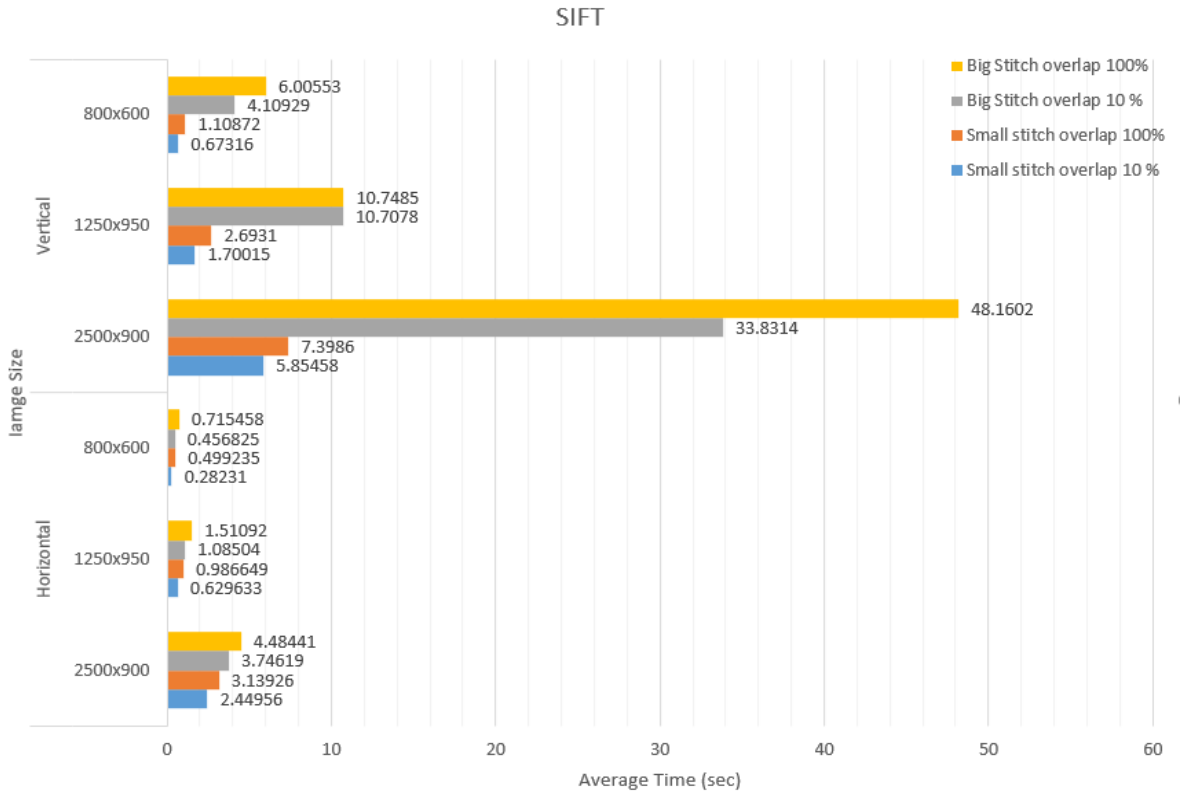


Figure 41 : Time analysis of SIFT algorithm, divided into vertical (upper half) and horizontal (lower half) stitching.

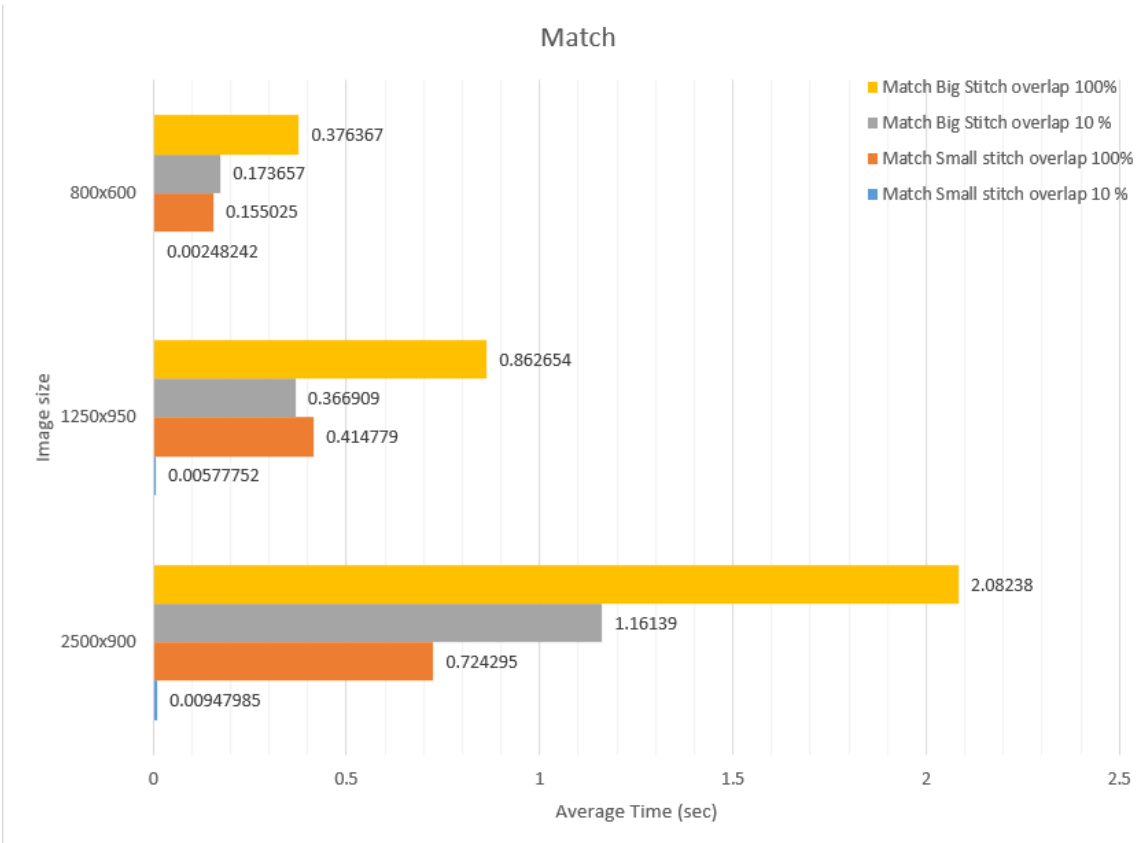


Figure 42 : Average time of completion of matching process for the horizontal stitches.

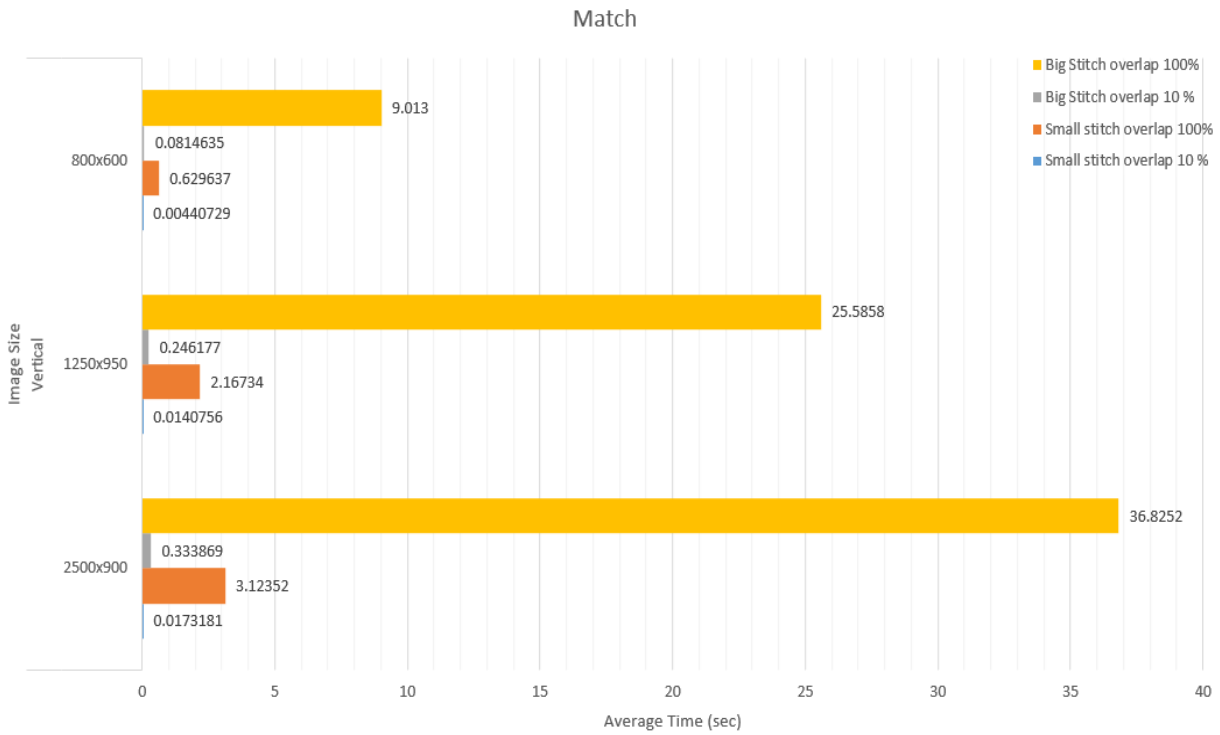


Figure 43 : Average time needed for completion of a vertical match.

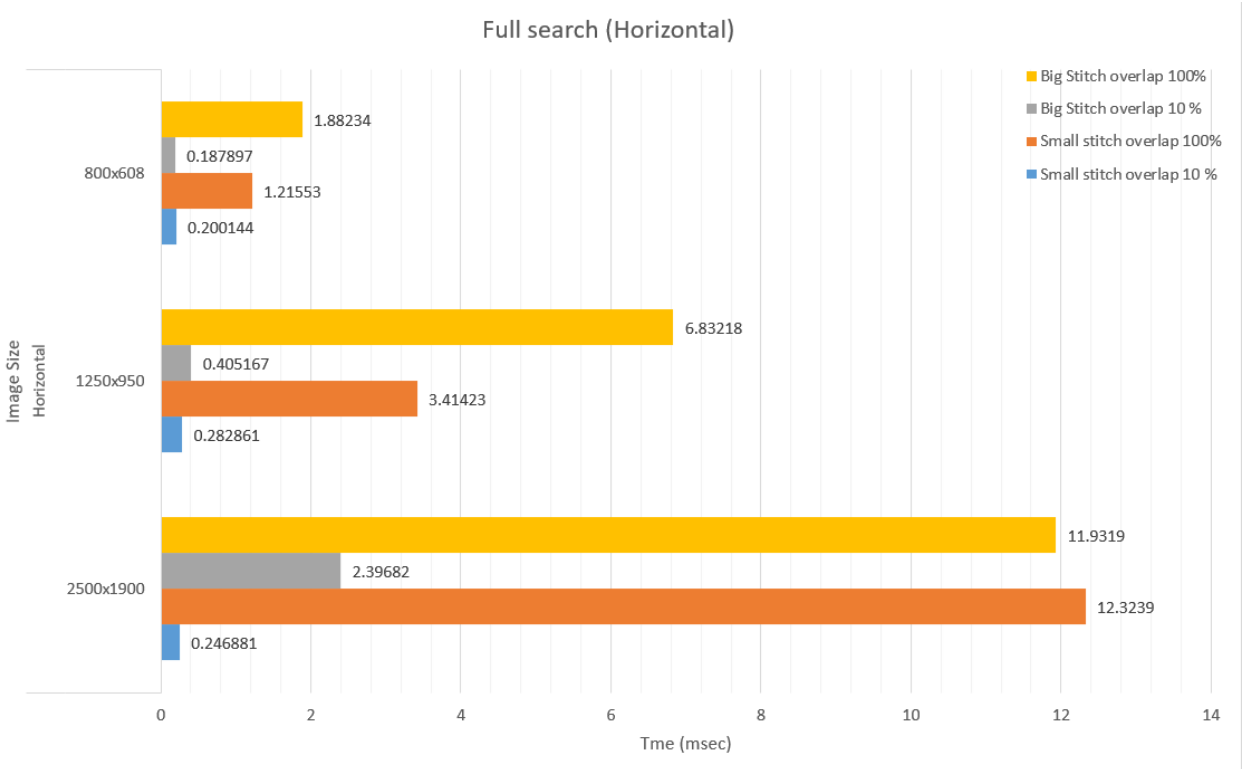


Figure 44 : Elapsed time for the full search method to complete (only horizontal)

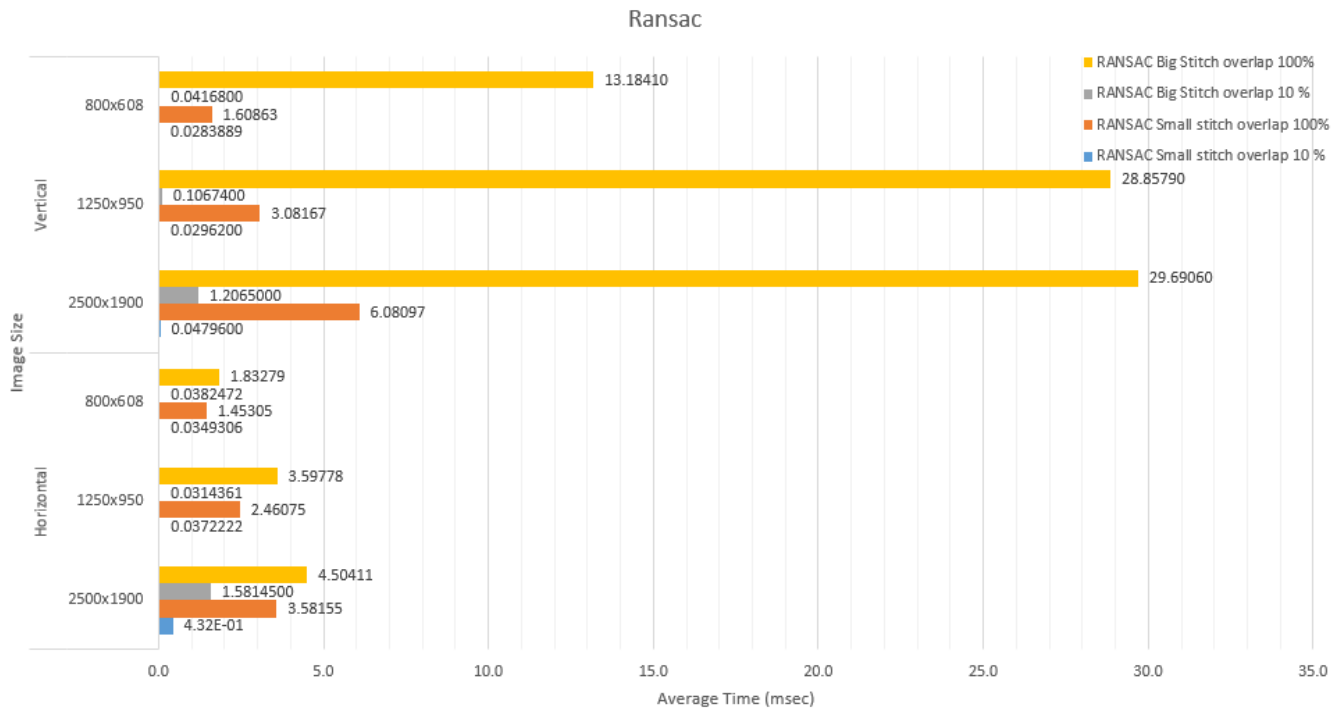


Figure 45 : Average performance of RANSAC

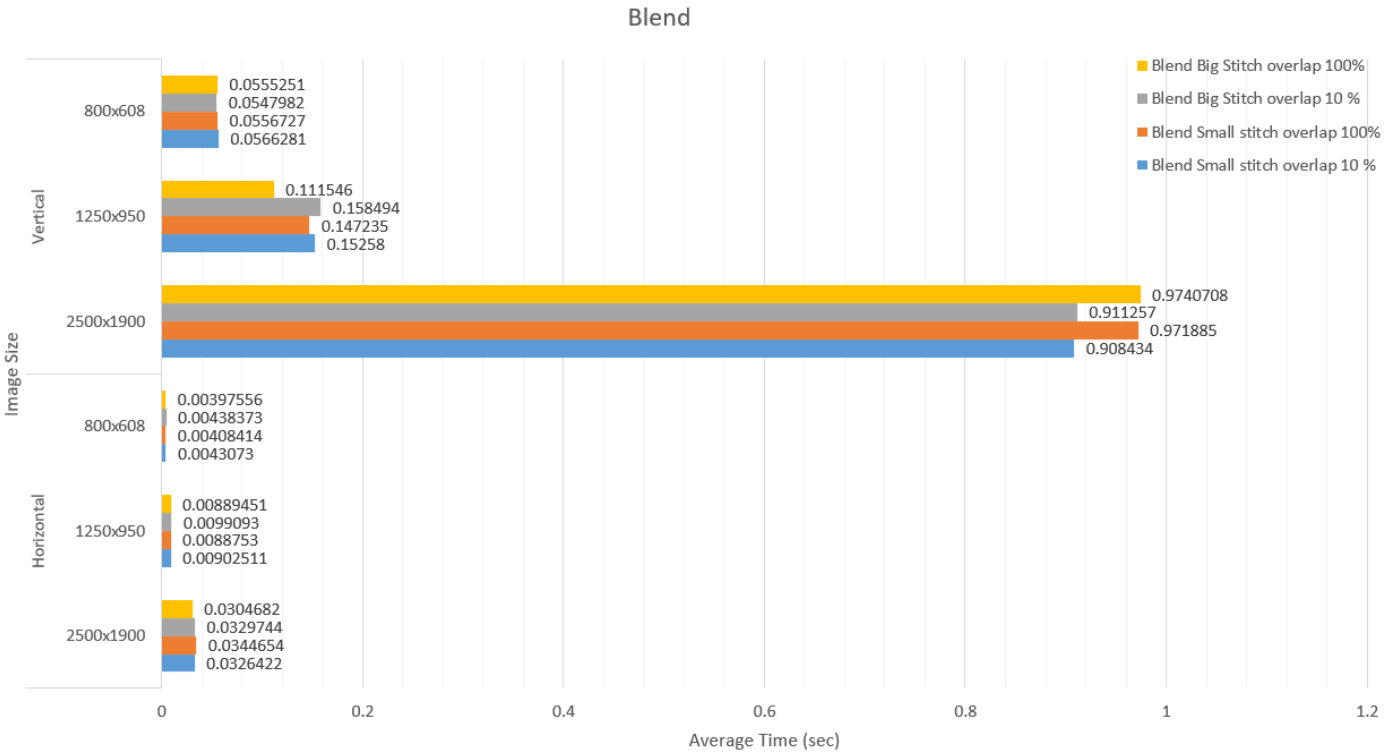


Figure 46 : The average time that will be needed to blend two images.

Table 1 Results of running full search and RASNAC methods.

		Small stitch		Big Stitch	
		overlap 10 %	overlap 100%	overlap 10 %	overlap 100%
Full Search	2500x1900	0.246881	12.3239	2.39682	11.9319
	1250x950	0.282861	3.41423	0.405167	6.83218
	800x608	0.200144	1.45305	0.187897	1.88234
RANSAC	2500x1900	0.0432	3.58155	1.5814500	4.50411
	1250x950	0.0372222	2.46075	0.0314361	3.59778
	800x608	0.0349306	1.21553	0.0382472	1.83279

5 Conclusions

The final chapter of this thesis will contain some conclusions drawn out during the implementation of the thesis and in conjunction with the results analyzed previously. Furthermore, even if the goal of this thesis has been achieved, there is always room for improvement, therefore there is also a section related to ideas for future work.

1.1 CONCLUSION

The work covered in this thesis was indended to create a system that would apply image stitching to pictures taken through the microscope, so that the whole sample will be available in one high-resolution image. The ultimate goal would be to use this system in automated microscopes to assist doctors and speed up the process of analyzing the samples. The big problem with using SIFT in such scenarios is the main disadvantage that the algorithm has, high computational cost, but by applying a number of techniques, and eliminating certain scenarios that are not expected to occur to the nature of the microscope (e.g., having images of successively different scale), that would otherwise increase the complexity of the problem and consequently increase the time needed for completion, the algorithm can be applied, providing with promising results. The important information to keep from the results is that using a combination of narrowing down the region of interest as well as rescaling the image in some cases can speed up the process of horizontal stitching up to 4 times. The results also showed that using a ROI of a smaller size provides a more significant advantage over just using the “small” stitch method on the full image. Aside from that, it seems to be always the better choice to combine all those techniques in order to minimize the time needed. Another vital thing to mention is the somewhat unexpected results of the comparison of RANSAC and Full Search methods. Although RANSAC always has better performance, since it will not need to search all the combinations of keypoints, due to the small size of the matching pair of keypoints and the inlier ratio.

1.2 FUTURE WORK

Although this thesis may be complete, there are numerous ideas that can extend this work. In mater of performance improvement, utilizing multithreading

programming and using the large number of cuda cores provided by GPUs, could potentially yield a significant performance gain from parallelism. Since the algorithm is not limited to only the scenario introduced in this thesis, another idea for future work would be to use the system for different kind of microscopic images. For example, it would be interesting to observe how differently the algorithm will perform when provided with images taken at different parts of the spectrum. Lastly, since an effort to analyze the contents of the images is already done in search of keypoints, it would be interesting analyze the image searching for cancerous cells, using some classification algorithm or utilizing neural networks trained to locate such cells in the sample.

Bibliography

[1] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features.," *The Proceedings of the Seventh IEEE International Conference on Computer Vision,* vol. 2, pp. 1150-1157, 1999.

[2] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," *Journal of Applied Statistics,* vol. 21, no. 2, pp. 225-270, 1994.

[3] K. Mikolajczyk, "Detection of Local Features Invariant to Affine Transformations," 2002.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, November 2004.

[5] M. a. L. D. Brown, "Invariant Features from Interest Point Groups," *Proceedings of the British Machine Vision Conference,* 2002.

[6] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[7] V. Hautamäki, I. Kärkkäinen and P. Fränti, "Outlier Detection Using k-Nearest Neighbour Graph," *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.,* vol. 3, pp. 430-433, 20 September 2004.

[8] S. E. Chen, "QuickTime VR – an image-based approach to virtual environment," *Computer Graphics (SIGGRAPH'95),* pp. 29-38, 1995.

[9] R. Szeliski, "Image mosaicing for tele-reality applications," *in IEEE Workshop on Applications of Computer Vision (WACV'94),* pp. 44-53, 1994.

[10] R. Szeliski and H. Y. Shum, "Creating full view panoramic image mosaics," *in Computer Graphics (SIGGRAPH'97 Proceedings),* pp. 251-258, 1997.

[11] R. Szeliski, Image Alignment and Stitching: A Tutorial, vol. 2, Foundations and Trends® in Computer Graphics and Vision, 2006, pp. 1-104.

[12] P. J. Burt and E. H. Adelson, "A multiresolution spline with applications to," *ACM Transactions on Graphics,* vol. 2, no. 4, pp. 217-236, 1983.

[13] P. Perez, M. Gangnet and A. Blake, "Poisson Image Editing," *ACM Transactions on Graphics,* vol. 22, no. 3, 2003.

[14] I. Rey-Otero and M. Delbracio, "Anatomy of the SIFT Method," *Image Processing On Line,* vol. 4, pp. 370--396, 2014.

[15] M. Z. Bonny and M. S. Uddin, "Feature-based image stitching algorithms," *2016 International Workshop on Computational Intelligence (IWCI),* pp. 198-203, 2016.

[16] E. Karami, S. Prasad and M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images," 2015.

[17] T. Kanade and B. D. Lucas, "An Iterative Image Registration Technique with an Applicatino to Stereo Vision," *Proceedings of Imaging Understanding Workshop,* pp. 121-130, 1981.

[18] J. J. Koenderink, "The structure of images," *Biological Cybernetics,* vol. 50, pp. 363-396, 1984.