Enhancement of the Fynbos Leaf Optical Recognition Application (FLORA-E)

by

Roy Makumborenga

BSc (Elec. Eng.)

Submitted to the University of Cape Town in fulfilment of the requirements

for the degree

Master of Science

Supervisor:

Dr Simon Winberg

Department of Electrical Engineering

Faculty of Engineering and the Built Environment

UNIVERSITY OF CAPE TOWN

December 2020

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signature of Author: . . Signed by candidate . . .

Roy Makumborenga

Date: 11 August 2020

Abstract

Object perception, classification and similarity discernment are relatively effortless tasks in humans. The exact method by which the brain achieves these is not yet fully understood. Identification, classification and similarity inference are currently nontrivial tasks for machine learning enabled platforms, even more so for ones operating in real time applications. This dissertation conducted research on the use of machine learning algorithms in object identification and classification by designing and developing an artificially intelligent Fynbos Leaf Optical Recognition Application (FLORA) platform.

Previous versions of FLORA (versions A through D) were designed to recognise Proteaceae fynbos leaves by extracting six digital morphological features, then using the k-nearest neighbour (k-NN) algorithm for classification, yielding an 86.6% accuracy. The methods utilised in FLORA-A to -D are ineffective when attempting to classify irregular structured objects with high variability, such as stems and leafy stems. A redesign of the classification algorithms in the latest version, FLORA-E, was therefore necessary to cater for irregular fynbos stems. Numerous algorithms and techniques are available that can be used to achieve this objective. Keypoint matching, moments analysis and image hashing are the three techniques which were investigated in this thesis for suitability in achieving fynbos stem and leaf classification. These techniques form active areas of research within the field of image processing and were chosen because of their affine transformation invariance and low computational complexity, making them suitable for real time classification applications.

The resulting classification solution, designed from experimentation on the three techniques under investigation, is a keypoint matching – Hu moment hybrid algorithm who`s output is a similarity index (SI) score that is used to return a ranked list of potential matches. The algorithm showed a relatively high degree of match accuracy when run on both regular (leaves) and irregular (stems) objects. The algorithm successfully achieved a top 5 match rate of 76% for stems, 86% for leaves and 81% overall when tested using a database of 24 fynbos species (predominantly from the Proteaceae family), where each species had approximately 50 sample images. Experimental results show that Hu moment and keypoint classifiers are ideal for real time applications because of their fast-matching capabilities. This allowed the resulting hybrid algorithm to achieve a nominal computation time of ~0.78s per sample on the test apparatus setup for this thesis.

The scientific objective of this thesis was to build an artificially intelligent platform capable of correctly classifying fynbos flora by conducting research on object identification and classification algorithms. However, the core driving factor is rooted in the need to promote conservation in the Cape Floristic Region (CFR). The FLORA project is an example of how science and technology can be used as effective tools in aiding conservation and environmental awareness efforts. The FLORA platform can also be a useful tool for professional botanists, conservationists and fynbos enthusiasts by giving them access to an indexed and readily available digital catalogue of fynbos species across the CFR.

Acknowledgements

I would like to thank the following for their help during the course of this project.

The Makumborenga family; my mother: Thank you for always believing in me and constantly reminding me that I can do anything if I put my mind to it.

My supervisor, Dr Simon Winberg: Your guidance during this project was insightful and invaluable. I count myself lucky to have had the privilege of sharing a coffee with you.

Nomenclature

CFR	Cape Floristic Region
API	Application Programming Interface
FFT	Fast Fourier Transform
DCT	Discrete Cosine Transform
4IR	Fourth Industrial Revolution
UX	User Experience
UI	User Interface
SI	Similarity Index
ROI	Region of interest
AI	Artificial Intelligence
DLL	Dynamic Link Library
PNG	Portable Network Graphics
RAM	Random Access Memory
FLANN	Fast Library for Approximate Nearest Neighbours
MVC	Model View Controller

Contents

List of Figures

List of Tables

1 Introduction

The Fynbos Leaf Optical Recognition Application Enhancement (FLORA-E) project is a continuation of the FLORA project initiated in 2011 [1]. Its aim is to build a high accuracy, AI enabled platform for the recognition of FYNBOS species that are found throughout the Cape Floristic Region (CFR) [2].

Apart from its alignment with global trends arising from the fourth industrial revolution, the FLORA platform can be used as a tool for educational and conservationist purposes. The Fynbos biome is a very delicately balanced ecosystem. The unnatural invasion of alien plant species and disruptive modern-day human activities have become a constant threat to the continued survival of many fynbos species across the Cape floristic region [3, 4]. To the conservationist and botanist, FLORA can be a useful tool documenting the unique vegetation of the CFR. A detailed database of fynbos species continuously updated is invaluable to conservation efforts and initiatives in the region.

This primary focus of this thesis is the design and development of an object identification and classification platform to be used in classifying fynbos flora stems and leaves.

1.1 Background

1.1.1 CFR Plant Diversity

Fynbos is the term used to refer to the distinct vegetation of South Africa`s mountainous Cape region [2, 5, 6]. It is part of the Cape Floral Kingdom, constituting approximately 80% of the total species that make up the kingdom, with over half of these species endemic to the Cape region [7, 8].

Figure 1.1. Some well-known FYNBOS species. Left: The bright red coloured Common Sunshine Cone bush (Leucadendron Salignum). Top: The Rooibos shrub Aspalathus Linearis. Bottom: The scented honey daisy (Euryops Virgineus)

The aim of the FLORA project is to build a platform capable of accurately identifying the various fynbos species that make up the Cape Floristic Region (CFR), also now commonly referred to as the Cape Core Subregion (CCR) by several scholars [2].

1.1.2 Taxonomical Classification in Botany

An understanding of botanical taxonomical classification is important to compare the accuracy of the FLORA platform with established and scientifically accepted methods of plant classification. Classification of plants into botanical groupings is a complex, well-coordinated process. The Linnaeus system, which is currently used for the taxonomic classification of biological organisms, was created by Carl Linnaeus in 1735 [5, 9]. Carl Linnaeus is also responsible for the binomial nomenclature used today for naming species [9].

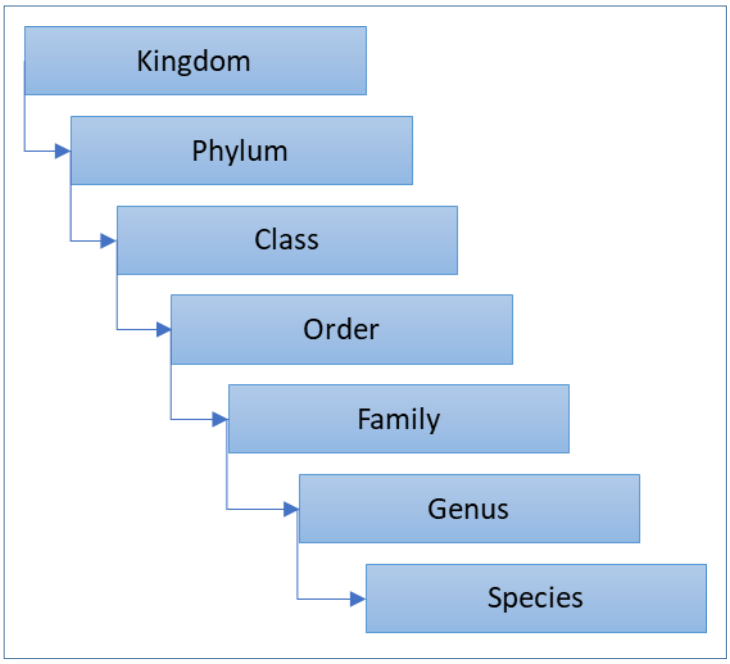


Figure 1.2.Taxonomic ranks used in the Linnaeus biological classification system.

The complexity of the characteristics used to group plants increases down the levels from the kingdom to the species classification levels. Plants, for instance, are grouped into Phylum (Divisions) depending on whether they produce spores or seeds. In contrast, lower lying groupings would look at characteristics such as the lifecycle of the plant, plant sexual expression (Monecious, Diecious, Hermaphroditic) or the climatic adaptation of the plant [5, 6]. Characteristics such as colour, smell, seed structure as well as in-depth biological analysis would all be considered in the lower classification levels [7, 8, 10].

1.2 Project Versioning and Terminology

The FLORA project was started in 2011 and is being continuously refined and upgraded [1, 11, 12]. The previous versions of the application all focused on identification and classification of fynbos leaves. Below is the list of previous versions along with the aims and outcomes achieved with each version.

a) **FLORA-A**: Primarily a MATLAB implementation of the underlying image processing and classification techniques. Work conducted by S. Katz [1] laid the foundation for the image processing and classification algorithms used by the FLORA application.
b) **FLORA-B** : The core FLORA methods and functions where reimplemented in Python [11].
c) **FLORA-C** : This version reimplemented the FLORA core from Python to C/C++ [13].
d) **FLORA-D**: Frontend components where redesigned using the Django python framework. An attempt to GPU accelerate FLORA was made [12, 13].

1.3 Problem Description

Past FLORA projects have already achieved a high degree of accuracy with leaf matching. FLORA-D achieved a matching accuracy of 86.6% when tested with Proteaceae leaves [1, p. iv]. The FLORA-A to -D test datasets were however very small and not suitable for real-world application. The descriptors and classification techniques used in previous FLORA versions are also not suitable for classification of leafy-stemmed or grass-like fynbos species such as those found in Ericaceae and Restionaceae families.

1.4 Motivation and Objectives

The motivating factors behind this project gave rise to a set of relatively well-defined objectives, both of which shall be discussed in detail below.

1.4.1 Motivation

The FLORA project is concerned with the building of an artificially intelligent platform for use in the identification of different species of fynbos across the fynbos biome using image processing and machine learning techniques. The fynbos biome constitutes

82.5% of the CFR with over 68.3% of the CFR species being endemic to the region [2, p. 10]. Standardised, readily available documentation and information on fynbos species across the CFR is important for botanists and conservationists. Instant identification of species can help facilitate the rapid cataloguing of endangered fynbos species. With continual extension of the fynbos image dataset, an important output of the platform is an indexed database of known species, their distribution across the biome as well as their endangered status. On top of being an awareness and conservation tool, the FLORA platform can aid further research in image processing techniques and machine learning algorithms useful in the identification and classification of regular and irregular objects.

1.4.2 Objectives

The objective of the project is to build an artificially aware object recognition and classification platform capable of identifying and classifying species from input images. The objectives are split into primary and secondary, primary objectives being the core points of investigation critical to the project's development path.

Primary Objectives

The key objectives this thesis investigates are listed below.

i. Critically evaluate available literature on object similarity and classification techniques in the fields of computer vision, image processing and artificial intelligence focusing on keypoint matching, Hu moments and hashing.
ii. Investigate similarity and classification techniques with respect to the projects use cases, i.e. on leaves and stems, to ascertain pros, cons and limitations for each technique in each use case.
iii. Conceptualise and develop an affine invariant classification algorithm capable of botanical family or genus level recognition in each of the use cases mentioned above.
iv. Expand the recognition capabilities of FLORA to encompass classification of not only regular shaped leaves but irregular structured stems as well. FLORA-E will therefore attempt to equip the application with the capability to identify broad leafed species such as Proteaceae fynbos, leafy stemmed species such as Ericaceae fynbos as well as grassy fynbos such as species in the Restionaceae fynbos family.

Secondary Objectives

The secondary objective of this thesis was the design and development of a web-based portal for the FLORA backend core.

1.5 Terms of Reference – Project Requirements

A list of requirements for the FLORA-E platform was compiled which were derived from the objectives outlined in Section 1.4.2. These requirements are listed below.

R1. System should be accessible to end users via an appropriate interface.

R2. System must recognise leaves.

R3. System must recognise stems.

R4. System must produce list of potential matches for a sample.

R5. System must be easy to train.

R6. Image dataset must be properly indexed.

R7. Computation time for matches must be enough for real time application.

R8. DLL deployment of FLORA backend core.

With the system requirements defined, a list of functionalities required from the system is compiled. The functionality will form the basis of system tests conducted to ensure the system is compliant with the set objectives. Both front-end and back-end system functionalities are considered.

F1. System training of in-memory classifiers.

F2. System training of file-based classifiers.

F3. Allow uploading image from front end UI for comparison.

F4. Classify stems.

F5. Classify leaves.

F6. Return list of potential matches with corresponding images in under 25 seconds.

F7. Realtime computation of matches.

F8. System should be cloud deployable.

Table 1.1 shows the sections of this thesis in which the above functionality and requirements were tested or demonstrated.

Table 1.1. Tests conducted to evaluate system requirements and functionality.

Sections	**Description**	**Functions Checked**	**Requirements Tested**
4.1.2, 4.2, 4.3	In-memory classifier training	F1, F2	R3, R4, R5
4.1.2, 4.2, 4.3	File based classifier training	F1, F2	R3, R4, R5
4.1.3, 4.2, 4.3, 5	Check system classifies leaves	F1, F2, F5, F6	R2, R4, R5, R6
4.1.3, 4.2, 4.3, 5	Check system classifies stems	F1, F2, F4, F6	R3, R4, R5, R6
4.4, 5	Check system returns ranked potential matches list	F6	R2, R3, R4, R6
5	Qualitative and quantitative check of returned match results	F4, F5, F6	R2, R3, R4
4.1.3, 4.2.3, 4.2.4, 4.4.1	Computational time check	F6, F7	R1, R7
5	Front-end results check	F3, F4, F5, F6, F7, F8	R2, R3, R7, R8

1.6 Scope and Limitations

The scope of this thesis is limited to the investigation of efficient and accurate techniques and algorithms used in object identification and classification as well as the subsequent design and development of a platform capable of this identification and classification using fynbos species as the underlying dataset.

The following limitations were encountered during the project:

- To the best of my knowledge, no digital dataset currently exists with a high quality and comprehensive list of fynbos species, more so one segmented into images of leaves and stems. This meant a limited dataset had to be collated and used for experimental procedures.
- Only techniques and algorithms capable of offering real time performance were considered as FLORA is a real-time application. These are keypoint matching, Hu moments and hashing.
- Due to order of priorities and time constraints, extraction of the region of interest (ROI) from a cluttered image was excluded from the objectives of this thesis. Efforts were concentrated on identifying and classifying an extracted ROI. Further versions of the app can look at ROI extraction techniques if needed.

1.7 Document Outline

Chapter 2 presents the review of various literature required to gain understanding of the problem description. It contains a discussion on the Cape Floristic Region and the taxonomical classification of fynbos species contained within it as well as the various image identification and classification techniques considered for FLORA-E.

Chapter 3 describes the various image similarity and classification algorithm tests conducted to ascertain suitability with respect to fynbos identification and classification. Focus is given to keypoint matching, hashing as well as continuous moment invariants. The chapter also looks at machine learning classification techniques and then concludes with a design discussion on feasible algorithms for encompassing the various researched techniques for use on stems and leaves.

Chapter 4 shows the results obtained from following the various experimental methodologies discussed in Chapter 3. This chapter provides empirical and qualitative analysis results which form the basis of the hybrid algorithm designed for FLORA-E.

Chapter 5 presents the hybrid algorithm used in FLORA-E for classification of stems and leaves. The custom algorithm used as well as how the different components work together makes up the bulk of the discussion in this chapter.

Chapter 6 provides conclusions on the work carried out throughout the dissertation as well as possible improvements that can be carried out on the project going forward.

2 Literature Review

Literature review provides a scientific basis for the techniques and algorithms under review in this thesis. The review starts off with a brief background on the Cape Floristic Region, its constituent biomes as well as the species that make up these biomes. The section will also explore species endemicity and the importance of conservation to the continued preservation of these species. An outline on scientific botanical classification methods is also given, which will offer parallels to the more automated and digital approaches discussed in this thesis.

An analysis of chosen keypoint matching algorithms is then conducted. The chosen algorithms were picked for their affine transformation invariance [14, 15] and potential use in real time applications. These are SIFT, SURF, ORB, KAZE and DAISY. This is then followed by similar analyses on image hashing algorithms as well as continuous and discrete moment invariants. This thesis introduces an object ranking metric called the similarity index (SI) which is derived from observations made from experimental analysis and is used to return an ordered list of potential matches. The derivation of a robust and reliable image similarity index (SI) score from test images relies heavily on algorithms that are affine invariant to a large degree, as well as resistant to the different noise forms. These include, but are not limited to, impulse noise (salt-and-pepper noise), amplifier noise (gaussian noise), shot noise, quantization noise (uniform noise), film grain, on-isotropic noise, multiplicative noise (speckle noise) and periodic noise [16]. Algorithms considered for this thesis therefore take into consideration, to various degrees, these different requirements.

2.1 The Cape Floristic Region

The Cape Floral Kingdom (Capensis), is one of the world's six floral kingdoms. The region encompasses an approximate area of 90 760km^2, making it the smallest of the floral kingdoms [2]. 68.3% of the over 9000 species in the floral kingdom are endemic to the region making it one of the world's most important biospheres [2, 17]. Table 2.1 below shows the distinct vegetation types that make up the CFR [2, p. 14].

Table 2.1. Life forms in the Cape flora.

Life Form	No. of species (% of flora)
Trees	220 (2.3)
Shrubs and subshrubs	$\pm$ 5000 (54.0)
Perennials	1035 (11.0)
Geophytes	1635 (17.2)
Graminoids	795 (8.4)
Annuals	612 (6.5)
Other	80 (0.6)
Total	9383 (100)

The Fynbos biome, which makes up 82.5% of the CFR [2], is a natural region that stretches from Vanrhynsdorp in the west to Grahamstown in the east, along a 200 kilometre-wide coastal strip in South Africa [17]. Table 2.2 shows the different biomes that make up the CFR [2, p. 7].

Table 2.2. Major vegetation types in the Core Cape Subregion.

Biome	Vegetation Type	Area 10^3 km^2 (% of CFR/CCR)	
Fynbos	Fynbos heathland	49.9 (55.0)	74.9 (82.5)
	Renosterveld	22.0 (24.2)	
	Strandveld thicket	3.0 (3.3)	
Succulent Karoo	Succulent shrubland	10.5 (12.0)	
Albany Thicket	Succulent thicket	2.9 (3.2)	
Afro temperate Forest	Evergreen forest	0.1 (0.1)	
Total		90.8 (97.8)	

Fynbos is mainly characterised by three main families, which all have an endemism of 91% and above. These are:

- Protea family (Proteaceae)
- Erica family (Ericaceae)
- Restio family (Restionaceae)

The most distinct fynbos species are members of these families. Examples include Protea Cynaroides commonly known as the king protea, the brightly coloured

pincushions like the yellow Leucospermum Prostratum as well as the large variety of the bell flower shaped species of the Erica family.

Figure 2.1. Left - The king protea. Right top - Erica Nudiflora. Right bottom - Yellow pincushions.

2.2 Fynbos Taxonomical Classification

As outlined in earlier sections, botanical classification of plant species is an intensive process which often requires scientific methods to distinguish species in a genus [7, 8, 10]. FLORA currently relies solely on visual data provided from a single perspective image for matching. With a comprehensive fynbos database containing a large dataset of species spanning the breadth of the genus, it becomes nearly impossible to distinguish one species from another structurally similar one using the classification techniques discussed in this thesis, which function solely on visual characteristics. Identification reliability is therefore limited to order or family and, at best, genus identification of the sample plant. This project focused on the collection of and experimentation on a diverse range of species across families (inter-family) as far as possible.

2.3 Object Recognition and Classification

Automatic object recognition and classification are active areas of research in image processing and machine learning. For this thesis focus was limited to three image classification techniques, namely keypoint matching, image hashing and moment invariants, which are described in detail below.

2.3.1 Keypoint Matching

The use of local interest points in image matching can be traced back to the early work conducted by Moravec in 1981 [18, p. 3] in which he conducted stereo matching using a corner detector. Since then, several researchers have published algorithms that showcase feature detection using corners, blobs, edges, junctions or lines. Each of these algorithms also use varying feature description techniques however all have a degree of affine invariance in common, a major advantage of keypoint matching algorithms. The algorithms explored for this thesis are the string-based Scale Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF), the binary based Oriented fast and Robust Brief (ORB) and Accelerated KAZE (AKAZE) as well as the wide baseline DAISY algorithm [14, 15, 18, 19].

SIFT, the most notable feature detector-descriptor algorithm was developed by D. Lowe in 2004 [18]. The SIFT algorithm has four steps [18, p. 1].

- The initial step involves detecting scale-space extrema using a Difference of Gaussian (DoG) function, an approximation of Laplacian of Gaussian (LoG).
- Keypoint localisation is then carried out which selects keypoints based on their stability.
- Orientations are then assigned to each keypoint using local image gradient directions making them orientation invariant.
- The last step is the computation of a distinct local image descriptor that is invariant to the remaining variations such as illumination or image viewpoint.

Computation of an image`s LoG is a two-step process which involves an initial convolution of the image with a smoothing gaussian to remove high frequency noise followed by convolution with a Laplacian filter kernel approximated from Equation 2.1.

$$\bar{\nabla}^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \qquad \textit{Eq 2.1}$$

The Laplacian operator calculates the second derivative of an image. This means in areas where the sample image has a constant intensity (where the intensity gradient is zero), the Laplacian response will be zero. The Laplacian will give a response in the vicinity of a change in intensity, making the Laplacian operator useful in edge detection. Two common Laplacian operator kernel approximations are shown in Figure 2.2 [20 p. 206, 21].

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 2.2. Laplacian kernel approximations. The kernel on the right is the Laplacian approximation that includes diagonal edges.

A Gaussian convolution along with the Laplacian operators shown in Figure 2.2 is necessary in practice because of the sensitivity of the second derivative approximation of the image carried out by the Laplacian kernel. Due to the associativity of convolution, practical implementations of the LoG involve convolving the Gaussian and Laplacian kernels first before convolving the result with the much larger dimensionality image. This significantly reduces the arithmetic complexity of the LoG operation because the LoG operation can be computed once using the derived result from the much smaller Gaussian and Laplacian kernels [18, p. 5]. An example of an image after convolving with the LoG is shown in Figure 2.3 [21].

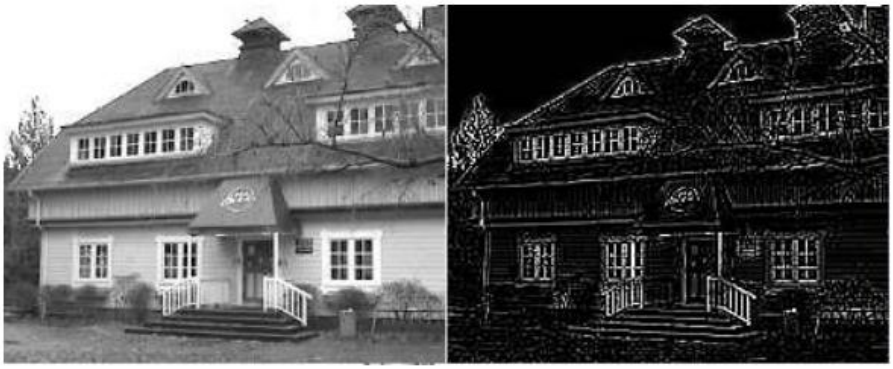


Figure 2.3. Example image showing result of convolving the image on the left with the LoG kernel.

SIFT descriptors are generated by taking a 16×16 region around detected keypoints, computing an 8-bin edge orientation histogram using 4×4 subregions, resulting in a 128-dimensional descriptor (16 cells × 8 orientations).

SURF, unlike SIFT, uses a Hessian matrix to approximate LoG as opposed to DoG for the detector. The descriptor used is a distribution of Haar-wavelet responses within the interest point neighbourhood. SURF also uses a reduced 64-dimensional descriptor resulting in less features needing computation [22, p. 4].

As outlined by E. Karami et al in [14] and E. Rublee in [19], Oriented fast and Robust Brief (ORB) is a fusion of the FAST key point detector and BRIEF descriptor with some modifications which include [19, p. 1]:

- The addition of a fast and accurate orientation component to FAST.
- The efficient computation of oriented BRIEF features.
- A learning method for de-correlating BRIEF features under rotational invariance, leading to better performance in nearest-neighbour applications.

E. Karami et al in [14] show that on average ORB computes matches faster than SURF and SIFT while maintaining reasonable accuracy. This makes ORB a realistic alternative to SIFT and SURF for real time applications and applications where computational power is limited.

Daisy is a wide baseline algorithm designed by E. Tola et al [15] which makes use of local region descriptors instead of correlation windows used by SIFT.

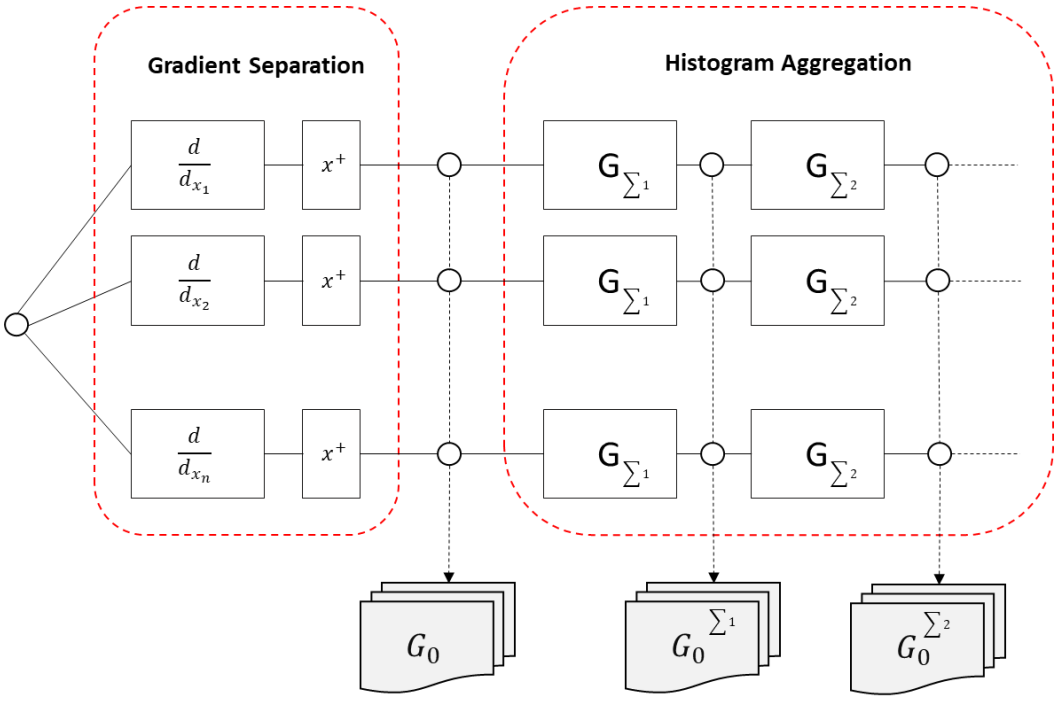


Figure 2.4. Computation of daisy descriptors [15, p. 3].

Computation of daisy descriptors [15, p. 3] involves first separating the gradients into different layers according to their orientation and then using Gaussian convolutions to perform the aggregation over different sized windows efficiently. Orientation maps from the original images are first computed, which are then convolved to obtain the convolved orientation maps $G_0^{\sum i}$. The values of the $G_0^{\sum i}$ correspond to the values in the SIFT bins. The $G_0^{\sum i}$ calculation can be obtained efficiently by chaining the convolutions. [15, p. 5].

Matching for these algorithms is then achieved using a variety of techniques, the most common being the calculation of L2-NORM (Euclidean distance) using the k-nearest neighbour (k-NN) algorithm. Further segregation of derived matches is achieved by implementing the nearest neighbour distance ratio test (also referred to in other literature as the second neighbour ratio test). This second neighbour ratio test compares the distance between the two closest matches. A match is considered viable if the distance between the two closest matches is greater than a certain threshold (D. Lowe in [18, p. 20] suggests match one`s distance from a sample should be less than 70% of the distance of match two). The second neighbour ratio test is discussed further in Section 3.4.3.

2.3.2 Image Hashing

Hashing involves the use of cryptographic hash functions to create a distinct digital signature. A key drawback of conventional hashing algorithms such as MD5 and SHA-1 [23] is that they are extremely sensitive to changes in the input data. Slight variations in the input message can lead to a completely different hash value [24, 25 p. 1]. This makes such functions non-ideal for image hashing purposes.

The process of obtaining a hash value from a perceptual image function is discussed by A. Hadmi et al in [26] in which they outline the four pipeline stages of a perceptual image hashing system.

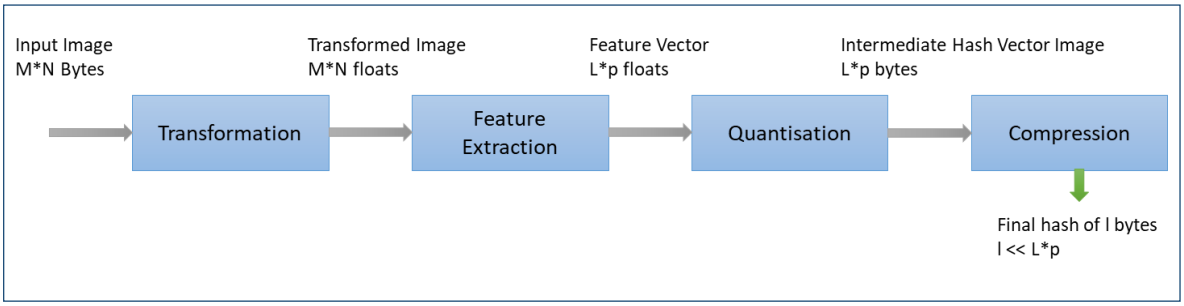


Figure 2.5. Four stage pipeline of a perceptual hashing system [26, p. 21].

The transformation stage takes an M×N image as input and performs spatial transformations such as colour transformation, smoothing or affine transformations. The transformations can also be frequency transformations such as the Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT) [26, p. 21].

Feature extraction varies for the various hashing techniques. Extraction in general will generate the feature vector of L features where L << M × N. Each feature can also contain P elements of type float meaning there will be L × P floats after extraction [26, p. 22].

The quantisation step results in the generation of an intermediate perceptual hash vector which contains L × P elements of type byte [26, p. 22]. Quantisation can take the form of setting a hash value to 0 or 1 depending on a determined threshold.

The final step of compression (possibly with encryption) involves the compression of the binary intermediate perceptual hash string into a short perceptual hash of fixed size of l bytes, where l << L × P [26, p. 22].

Several perceptual image hashing functions are currently available that can be used for generating distinct hash values from images. These include:

- Average hash
- PHash
- Block Mean hash
- Radial Variance hash
- Marr Hildreth hash
- Colour Moment hash

Average Hash

This is the least complex of the image hashing algorithms. The hashing process is discussed in-depth in [26–28] and follows the steps outlined below.

1) Remove high frequencies from the input image by reducing the size to 8x8 resulting in a total of 64 pixels.
2) Convert the image to grayscale thereby converting the three channel RGB values into single channel values ranging from 0-255.
3) Compute the mean of the 64 grayscale values.
4) Compare the 64 grayscale values to the mean. If the value is greater than the mean, set the value to 1 else set to 0.
5) Construct the binary hash by reading the values from the previous operation. This gives a 64-bit (8 byte) hash code. The order in which the values are read is not important if consistency in reading the values is maintained for all images.

PHash

The pHash function is similar to the Average hash function except that it uses the DCT to reduce the high frequencies in the image [27, 29 p. 30]. The 2D DCT image coefficient, $F(u,v)$, of an input N×M image with the intensity of the pixel in row i and column j denoted by $f(i,j)$ can be calculated using the below equation.

$$F(u,v) = \sqrt{\frac{2}{N}}\sqrt{\frac{2}{M}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \alpha_{ij} \,.\, cos\left[\frac{\pi u}{2N}(2i+1)\right] cos\left[\frac{\pi v}{2M}(2j+1)\right] f(i,j) \qquad \textit{Eq 2.2}$$

The theoretical background of how the DCT function works will not be covered in this thesis however extensive work has been conducted by numerous authors on this topic in [26], [25] and [29].

The result of the DCT step is that the pixels with low frequencies are now located in the upper left corner. The image can be cropped to 8×8 pixels starting from the top left location. With the image reduced to an 8×8 matrix, the succeeding steps follow the same procedure as the Average hash as indicated below.

1) Compute the mean of the 64 grayscale values.
2) Compare the 64 grayscale values to the mean. If the value is greater than the mean, set the value to 1 else set to 0.
3) Construct the binary hash by reading the values from the previous operation. This gives a 64-bit (8 byte) hash code. The order in which the values are read is not important as long as consistency in reading the values is maintained for all images.

Block Mean Hash

The block mean value based perceptual image hash function was proposed in 2006 by B. Yang et al in [30] in which they discuss four block mean methods [30, p. 2] which are summarised below.

Method 1

The first method follows the steps below:

a) Normalize the original image into pre-set sizes.
b) Divide the normalized image I into non-overlapping blocks $I_1, I_2, \ldots, I_N$, where N is the block number and equal to the length of the final hash value string.
c) Encrypt the indices of the block sequence $\{I_1, I_2, \ldots, I_N\}$ using a secret key K to obtain a block sequence with a new scanning order $\{I`_1, I`_2, \ldots, I`_N\}$.
d) Calculate the mean value sequence $\{M_1, M_2, \ldots, M_N\}$ from corresponding block sequence $\{I`_1, I`_2, \ldots, I`_N\}$ and obtain the median value M_d of this sequence as:

$$M_d = median\,(M_i) \qquad (i = 1,2 \ldots . N) \qquad \textit{Eq 2.3}$$

e) Normalize the mean value sequence into a binary form and obtain the hash values h as:

$$h_i = \begin{cases} 0 & M_i < M_d \\ 1 & M_i \geq M_d \end{cases} \qquad \textit{Eq 2.4}$$

Method 2

The second method is like method one with the slight variation that pixels are split into overlapping blocks. B. Yang et al specify that the degree of overlapping is set to half the size of a block [29, p. 28].

a) Normalize the original image into pre-set sizes.
b) Divide the normalized image I into overlapping blocks $I_1, I_2, \ldots, I_N$, where N is the block number and equal to the length of the final hash value string.
c) Encrypt the indices of the block sequence $\{I_1, I_2, \ldots, I_N\}$ using a secret key K to obtain a block sequence with a new scanning order $\{I`_1, I`_2, \ldots, I`_N\}$.
d) Calculate the mean value sequence $\{M_1, M_2, \ldots, M_N\}$ from corresponding block sequence $\{I`_1, I`_2, \ldots, I`_N\}$ and obtain the median value M_d of this sequence as:

$$M_d = median\,(M_i) \qquad (i = 1,2 \ldots . N) \qquad \textit{Eq 2.5}$$

e) Normalize the mean value sequence into a binary form and obtain the hash values h as:

$$h_i = \begin{cases} 0 & M_i < M_d \\ 1 & M_i \geq M_d \end{cases} \qquad \textit{Eq 2.6}$$

Method 3

The third method incorporates robustness to rotation by following the below steps:

a) Normalize the original image into pre-set sizes.
b) Divide the normalized image I into non-overlapping blocks $I_1, I_2, \ldots, I_N$, where N is the block number and equal to the length of the final hash value string.

c) Encrypt the indices of the block sequence $\{I_1, I_2, \ldots, I_N\}$ using a secret key K to obtain a block sequence with a new scanning order $\{I`_1, I`_2, \ldots, I`_N\}$.

d) Calculate the mean value sequence $\{M_1, M_2, \ldots, M_N\}$ from corresponding block sequence $\{I`_1, I`_2, \ldots, I`_N\}$ and obtain the median value M_d of this sequence as:

$$M_d = median\,(M_i) \qquad (i = 1,2 \ldots . N) \qquad \textit{Eq 2.7}$$

e) Rotate the matrix M by D degrees where M = $\{M_1, M_2, \ldots, M_N\}$ and D = {0, 15,30, … , 345}. Divide rotated matrix M_i (*i* = 1, 2, …, 24) into N blocks. Obtain the mean value sequence $\{M_{i1}, M_{i2}, \ldots, M_{iN}\}$ of each block and median value M_{di} of this sequence, which forms 24 groups of sequences.

Method 4

Method four is a combination of the overlapping method two and the block rotation method three. The image is divided into overlapping blocks then hash is calculated using the rotated mean values of the blocks as described in the method three.

a) Normalize the original image into pre-set sizes.

b) Divide the normalized image I into overlapping blocks $I_1, I_2, \ldots, I_N$, where N is the block number and equal to the length of the final hash value string.

c) Encrypt the indices of the block sequence $\{I_1, I_2, \ldots, I_N\}$ using a secret key K to obtain a block sequence with a new scanning order $\{I`_1, I`_2, \ldots, I`_N\}$.

d) Calculate the mean value sequence $\{M_1, M_2, \ldots, M_N\}$ from corresponding block sequence $\{I`_1, I`_2, \ldots, I`_N\}$ and obtain the median value M_d of this sequence as:

$$M_d = median\,(M_i) \qquad (i = 1,2 \ldots . N) \qquad \textit{Eq 2.8}$$

e) Rotate the matrix M by D degrees where M = $\{M_1, M_2, \ldots, M_N\}$ and D = {0, 15,30, … , 345}. Divide rotated matrix M_i (*i* = 1, 2, …, 24) into N blocks. Obtain the mean value sequence $\{M_{i1}, M_{i2}, \ldots, M_{iN}\}$ of each block and median value M_{di} of this sequence, which forms 24 groups of sequences.

Radial Variance Hash

The radial variance function was introduced by F. Lefebvre et al in [31]. The hash function is based off the Radon function postulated by J. Radon in [32]. The Radon transform is the integral transform consisting of the integral of a function over a straight line [29 p. 22, 33 p. 288]. To extend the Radon transform to discrete images, the line integral along d = x*cos α + y*sin α can be approximated by summing the pixels lying in a one pixel wide strip [29 p. 22, 33 p. 288]. The hash can be further improved by applying the DCT function to the radial variance vector [29, p. 26].

Marr Hildreth Hash

The Marr Hildreth based hash functions, unlike the radial variance function, calculate hash values based on image corner information [29 p. 22, 33 p. 289]. The Marr Hildreth edge detection operator was proposed by D. Marr and E. Hildreth in [20]. The Marr-Hildreth operator is a combination of a Gaussian filter (used to smooth the image) with a Laplacian (LoG).

Colour Moment Hash

The Colour Moment hash was suggested by Z. Tang et al in [34]. The Colour Moment hash is distinct from the other hashing algorithms in that it extracts hash values from an image`s hue, saturation and luminance (HSI) values and not from its grayscale constituent values [34, p. 645]. Colour Moment hash follows the following steps:

Figure 2.6. Colour moment hash computation process [35, p. 645].

Letting R, G, and B denote the red, green and blue components of a pixel, the conversion of an image into its hue (H), saturation (S) and luminance (I) components can be achieved using the equations below:

$$H = \begin{cases} \alpha & if\ B \leq G \\ 360 - \alpha & otherwise \end{cases} \quad \textit{Eq 2.9}$$

$$S = 1 - \frac{3}{(R+G+B)}[min(R,G,B)] \quad \textit{Eq 2.10}$$

$$I = \frac{1}{3(R+G+B)} \quad \textit{Eq 2.11}$$

The hash function then goes on to utilise invariant moments (first introduced by Hu in [35]) which are discussed in-depth in Section 2.3.3.

2.3.3 Moment Invariants

Moments are scalar quantities which, from a mathematical point of view, are projections of a function onto a polynomial basis [36]. The general 2-dimensional geometric moment of an image, M_{pq} of the (p+q) th order where p and q are non-negative integers is shown in Equation 2.12.

$$M_{pq} = \iint_{-\infty}^{\infty} x^p y^q\, f(x,y)\, dxdy \quad \textit{Eq 2.12}$$

The use of moment invariants for visual pattern recognition was introduced by M. Hu in 1962 [35]. Hu introduced six absolute orthogonal invariants and one skew orthogonal invariant derived from the first ten geometric central moments i.e. central moments up to the third order [37].

$$\emptyset_1 = \mu_{20} + \mu_{02} \qquad \textit{Eq 2.13}$$

$$\emptyset_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \qquad \textit{Eq 2.14}$$

$$\emptyset_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \qquad \textit{Eq 2.15}$$

$$\emptyset_4 = (\mu_{30} - \mu_{12})^2 + (\mu_{21} - \mu_{03})^2 \qquad \textit{Eq 2.16}$$

$$\emptyset_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} - \mu_{12})[(\mu_{30} - \mu_{12})^2 - 3(\mu_{21} - \mu_{03})^2]$$
$$+ (3\mu_{21} - \mu_{02})(\mu_{21} - \mu_{03})[3(\mu_{30} - \mu_{12})^2 - (\mu_{21} - \mu_{03})^2] \qquad \textit{Eq 2.17}$$

$$\emptyset_6 = (\mu_{20} - \mu_{02})[(\mu_{30} - \mu_{12})^2 - (\mu_{21} - \mu_{03})^2]$$
$$+4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} - \mu_{03}) \qquad \textit{Eq 2.18}$$

$$\emptyset_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} - \mu_{12})^2 - 3(\mu_{21} - \mu_{03})^2]$$
$$-(\mu_{30} - 3\mu_{12})(\mu_{21} - \mu_{03})[3(\mu_{30} - \mu_{12})^2 - (\mu_{21} - \mu_{03})^2] \qquad \textit{Eq 2.19}$$

Hu`s moments are invariant under scaling, translation and rotation transformations [35, 38]. Moments provide highly efficient local descriptors when used with continuous noise-free images [37]. L. Kotoulas and I. Andreadis in [37] analyse some of the different types of moments currently in use as depicted in the Table 2.3.

Table 2.3. Various continuous and discrete moments.

	Noise Sensitivity	**Computational Complexity**	**Reconstruction Accuracy**	**Native Invariance**
Moments of monomials				
Geometric moments	High	Low	N/A	No
Complex Moments	High	Low	N/A	Yes
Moments of continuous orthogonal basis				
Legandre	Low	High	Medium	Yes
Zernike	Low	Very High	High	Yes
Moments of discrete orthogonal basis				
Tschebichef	Low	Medium	Very High	No
Hahn	Low	Medium	Very High	No

As can be seen from Table 2.3, geometric moments (from which Hu moments are derived) have a very low noise tolerance however their computational complexity is low enough to allow for use in real time applications. Moments of continuous orthogonal basis offer the highest tolerance to noise and are natively invariant. However, these have a relatively higher computational power requirement than moments of monomials.

3 Methodology

This thesis followed a phased approach in the design of a reliable stem and leaf classification algorithm capable of real time application for use in the FLORA platform. The 4 phases of this dissertation are shown in Figure 3.1.

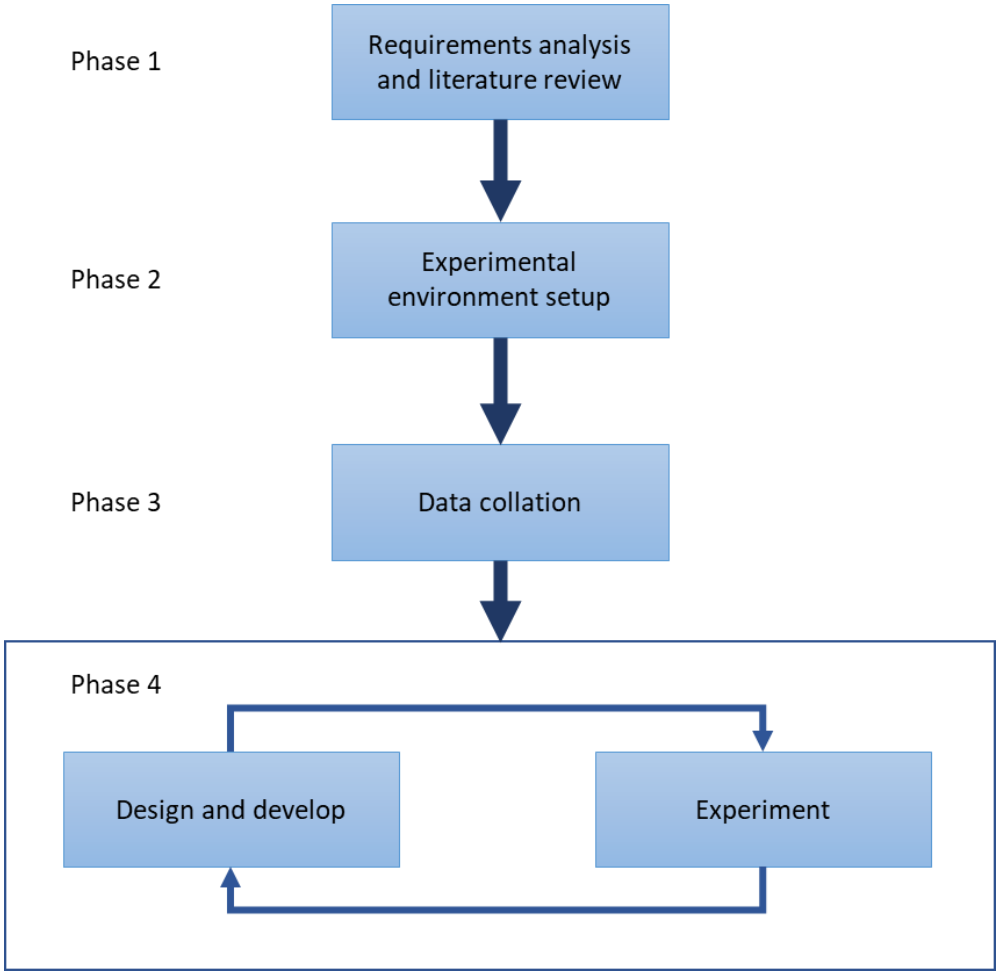


Figure 3.1. Project phases.

3.1 Phase 1: Requirements Analysis and Literature Review

The design of FLORA-E followed defining a set of requirements for the project. These requirements are elaborated in Chapter 1 along with the motivation for the project. Once the requirements were established, literature review (Chapter 2) was conducted to investigate the various techniques and algorithms available that could be used to fulfil the requirements. Several image matching techniques were considered in the design of FLORA-E, each with its own pros and cons. As stated in the literature review, this thesis focused on three techniques and their usefulness in deriving a reliable

image similarity index with respect to fynbos plants. The three techniques are keypoint matching, Hu moments and image hashing, chosen for either one or all their experimentally proven robustness, efficiency and resistance to affine transformations and noise.

3.2 Phase 2: Experimental Environment Setup

This phase involved the initial setup of the software and hardware required during the experimental procedure. The image processing library, OpenCV, was required for the experiments conducted in this thesis. The following equipment was used.

- 2.7 GHz 6th generation Intel Core i7, 32 GB RAM machine with NVIDIA Quadro M3000M graphics processor
- OpenCV 4.0.1
- Visual Studio 2019 Community
- Cuda 10.0.130_411.31_win10 extension for Visual studio

3.3 Phase 3: Data Collation and Preparation

At the time of writing this thesis, to the best of my knowledge, no comprehensive digital database of fynbos stems or leaves existed. S. Privett in [39] makes an attempt to compile a digital archive, but it is neither comprehensive nor of the required quality and quantity. Due to the non-availability of such databases, a data collection phase was included in the project which involved the time-consuming process of manual collection of specimen images. The collected dataset is primarily from the Proteaceae family with Restionacae species mimicked by leafy stemmed unnamed samples.

Collection locations where limited to the Western Cape region of South Africa and included:

- Kirstenbosch botanical gardens.
- Table Mountain nature reserve.
- Stellenbosch reserve.
- University of Cape Town school of botany.

Table 3.1. Table of collected samples. More stem samples were collected than leaf samples because previous FLORA versions concentrated on leaf analysis. The focus of FLORA-E is primarily stem analysis.

Sample ID	Scientific Name	Common Name	No. of Stem Samples	No. of Leaf Samples
1	Unnamed Species	-	48	-
2	Unnamed Species	-	-	43
3	Unnamed Species	-	51	-
4	Unnamed Species	-	51	-
5	Unnamed Species	-	53	-
6	Unnamed Species	-	-	56
7	Unnamed Species	-	45	-
8	Unnamed Species	-	40	-
9	Unnamed Species	-	42	-
10	Unnamed Species	-	53	-
11	Unnamed Species	-	42	-
12	Unnamed Species	-	40	-
13	Unnamed Species	-	40	-
14	Unnamed Species	-	32	-
15	Leucospermum Praecox	Albertina Sand Fynbos	50	-
16	Mimetes Cucullatus	Kogelberg Sandstone	51	-
17	Leocospermum Glabrum	Pincushion	50	33
18	Leucospermum	Thompsons Phoenix	49	40

19	Protea Lanceolata	Lance-Leaf Sugarbush	-	39
20	Leucodendron Teritifolium	Waterbossie	46	-
21	Serruria Fucifolia	Northern Spiderhead	47	-
22	Leucospermum Reflexum	Mountain Fynbos	45	
23	Leucospermum Grandiflorum	Grey leaf fountain	50	50
24	Leucospermum Formosum	Silver leaf wheel	46	-

Figure 3.2. Collected fynbos samples. Left: A Pincushion stem sample (Leocospermum Glabrum - Sample 17). Right: Albertina Sand Fynbos leaf sample (Leucospermum Praecox - Sample 15).

Limitations to the number of species that could be collected within the geographical area in reach resulted in supplementation of the collected fynbos species dataset with non-fynbos species, all of which however where still structurally like the fynbos species under investigation. Doing so added much larger variability between species allowing for broader testing of the system.

As mentioned in Section 1.6, ROI extraction from a cluttered background was not considered as a primary objective of this dissertation. Removal of unwanted background was therefore conducted manually using a pure white background during the image capturing process. To further improve on efficiency during experimentation, when conducting hashing analysis for instance which relies on the generation of a hash key based on existing pixels in the image, the images are further reduced to PNG files by extracting the ROI from the white background rendering the white background transparent. This step ensures experimental results are only influenced by pixels in the ROI and not by background clutter.

3.4 Phase 4: Design, Development and Experimentation Iteration

An iterative approach was used for the Design-Develop-Experiment process. This allowed refinement of the designed algorithm in response to results obtained from experimental procedures.

3.4.1 Data Splitting

For data splitting required in training k-NN or Hamming distance based classifiers, validation and testing followed on the work conducted by Reitermanova in [40] and Xu et al in [41]. It is important to avoid over-fitting data to the model which leads to results that do not reflect the true performance of the system. Reitermanova shows that for datasets with low dimensionality, any of the data splitting methods can be made use of [40, p. 6]. Hu moments, with 7 descriptors, fall in this category. High dimensional datasets however (such as those derived from keypoint matching and hashing) are best split using more complex methods such as cluster sampling or stratified sampling [40, p. 35]. The current dataset is already split into stems and leaf samples (strata) therefore splitting was done using k-fold cross validation with stratified sampling [40 p. 2, 41 p. 3]. The FLORA dataset is relatively small hence using the k-fold strategy would introduce a lot more variance than other strategies, like hold-out for example.

Data splitting was the first step in the experimental procedure to determine the most effective algorithms and parameters to use for descriptor extraction. Initial splitting utilised image to image analysis for all the classification algorithms considered. Through iterative design and development, this was eventually refined to use the classifier-based algorithms. The relative match accuracy between folds was more important when running the k-fold cross validation strategy than the overall accuracy of the utilised algorithms therefore the choice of specific descriptor extraction algorithm

utilised for each of the three classification techniques was primarily influenced by its ease of implementation. The ORB algorithm was used for descriptor extraction for keypoint classification, chosen for its binary descriptors which are faster to compute than string descriptors. The Hu moments algorithm was used for moment descriptor extraction and the binary descriptors of the Average hash algorithm were utilised for hashing.

3.4.2 Classifier Training

The classifier based experimental procedures described in this thesis all follow the same methodology which splits their design into training and test components. The training component involves the initial extraction of descriptors from the test dataset and the subsequent training of either an N vectored k-NN classifier or an N vectored Hamming distance classifier, where N is the bin size of the specific descriptor.

The training methodology used is outlined in the flowchart depicted in Figure 3.4. The result of a successful training phase is the generation of trained matching k-NN or Hamming distance classifiers for each of the algorithms under investigation.

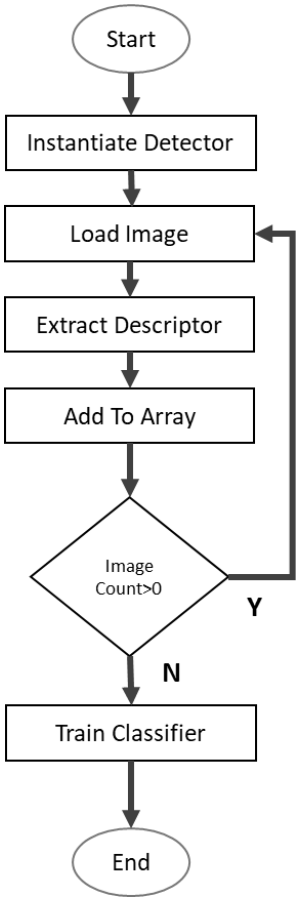


Figure 3.3 Process flowchart for the training process employed for the k-NN and Hamming distance classifiers. Descriptors are extracted from training images and stored in a temporary array. Once all image descriptors have been extracted, the classifier is trained and stored in memory or to a file.

3.4.3 Keypoint Matching

Section 2.3.1 of the literature review chapter showed how D. Lowe (SIFT) [18], H. Bay et al (SURF) [22] and E. Rublee et al (ORB) [19] all postulated keypoint matching algorithms that offer a measure of similarity between objects in an image. In these postulations, the authors conducted experiments on the same object or image scene transformed in various ways. This section will outline the design of a keypoint matching based algorithm for use in identification and classification while also proposing tests to ascertain the effectiveness and accuracy of various keypoint matching algorithms on geometrically similar and regular objects (leaves) as well as geometrically similar but irregular objects (leafy stems).

Table 3.2. Table showing the various keypoint algorithms, the descriptor type (binary or string), the bin size and the detection method for the descriptor.

	SIFT	**SURF(64D)**	**ORB**	**KAZE**	**AKAZE**	**DAISY**
Detection Method	Blobs	Blobs	Corners	Blobs	Corners	Blobs
Descriptor Type	String	String	Binary	String	Binary	String
Descriptor Bin Size	128 floats	64 floats	32 bytes	128 floats	486 bits	-

Algorithm testing will follow some of the procedures suggested by E. Karami et al in [14]. With regards to keypoint matching, this thesis explores three testing metrics to shed light on various parameters of interest which include:

1) Speed performance and efficiency of the algorithm.
2) Accuracy.
3) Repeatability and clustering efficiency.

The two testing procedures are direct image-to-image comparison and classifier-based comparison. Each are discussed in detail below.

Image-to-Image Analysis

This testing procedure is essential for creating a performance benchmark for the other two testing procedures as it is the simplest and least optimised. Testing procedure follows the process illustrated in Figure 3.4 and utilises the in-built OpenCV detect and compute functions along with a custom keypoint segregation function.

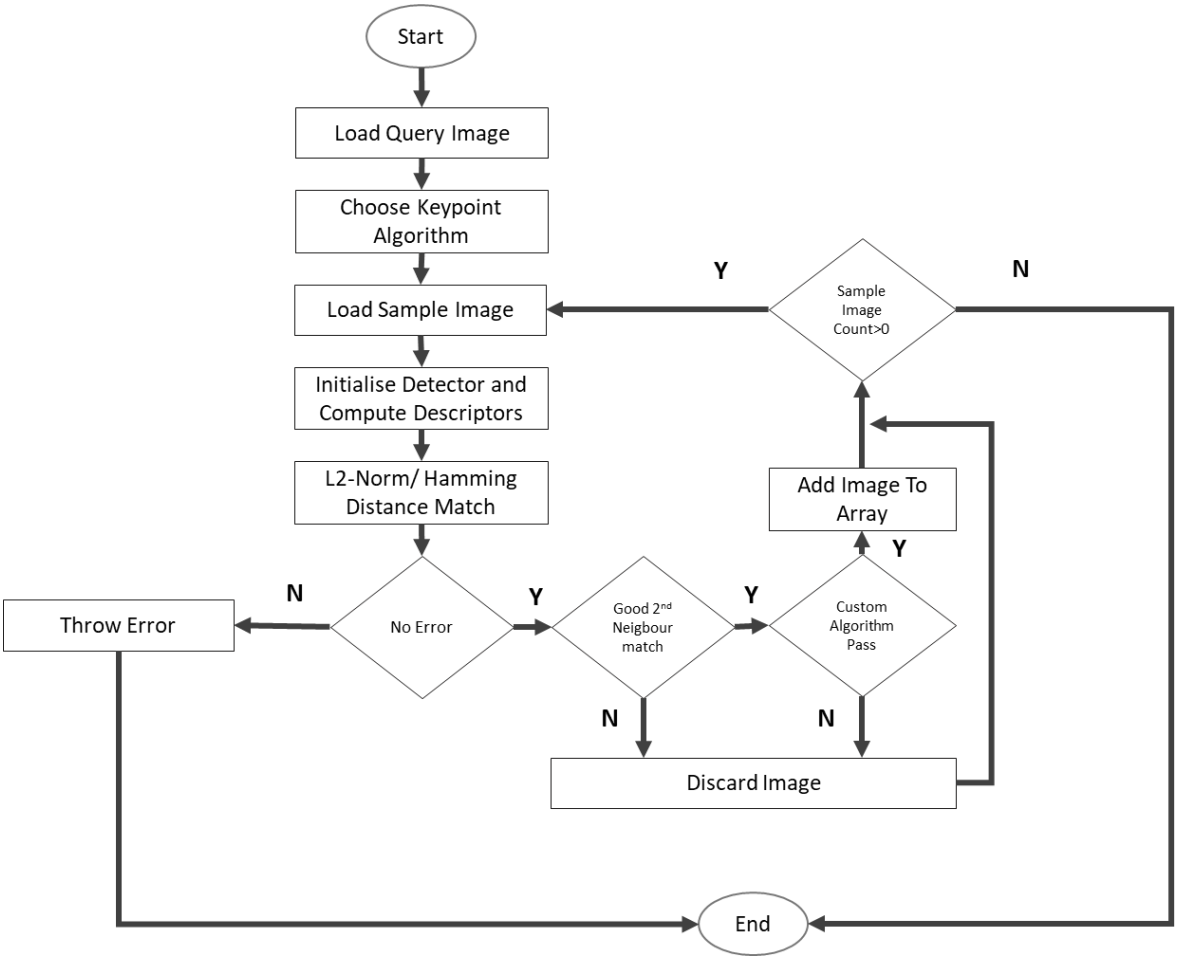


Figure 3.4. Image-to-image analysis testing procedure.

Image-to-image comparison involves taking a test image and iterating through the raw training image dataset, detecting and computing descriptors in each iteration and ascertaining feasible descriptor matches from the result. The designed custom matching algorithm makes use of the following key consideration:

- An image is considered a viable match if the variance in keypoints of the image with the larger array of keypoints is within 30% of the smaller sized keypoint array. i.e.

$$\left({}^{smaller\ keypoint\ array\ size}/_{larger\ keypoint\ array\ size}\right) \geq 0.70 \qquad \textit{Eq 3.1}$$

  Keypoint array size is not a definitive indicator of similarity but, when used in conjunction with other indicators, is a good segregation technique. A

comprehensive dataset benefits immensely from this test as the size of such a dataset would slow down the performance of the system if such a check is not in place. Experimental results in later sections of this thesis will show how this check, like the second neighbour ratio test, eliminates a lot of false positives while preserving a good majority of the valid matches.

- A match is considered viable if there is a 1% match rate or greater between the test image and the query image. Experimental results presented in this thesis will also show how the match rate between irregular and geometrically similar but different objects is substantially lower than when matching an object with its modified version (sheared, rotated or affine transformed) however the relative match rates are still useful enough to provide insight for matching purposes. Again, this test is very useful in segregating non-viable matches. The chosen minimum percentage match rate is also heavily dependent on the size of the dataset. A larger, more comprehensive dataset will have much higher match rates and therefore the minimum match rate is expected to be much higher. The minimum accepted match rate can be determined through experimentation or, much quicker, through extrapolation.

The second neighbour test suggested by D. Lowe et al in [18] is utilised to carry out initial filtering of feasible matches. D. Lowe suggested the following ratio in order to consider a match distinctly viable for classification purposes where d_1 is the distance of the first match from the sample descriptor and d_2 is the distance of the second match from the sample descriptor [18, p. 20].

$$\left({}^{d_1}/_{d_2}\right) < 0.70 \qquad \textit{Eq 3.2}$$

Experimental investigation showed that a ratio of 0.6 gives the best results with the test and training data at hand.

Classifier Based Comparison

The classifier-based classification approach is similar in many ways to the image-to-image comparison but with a few changes that have a significant effect on the performance, usability and reliability of the matching process as outlined in the discussion that follows and validated by the results obtained in Chapter 4.

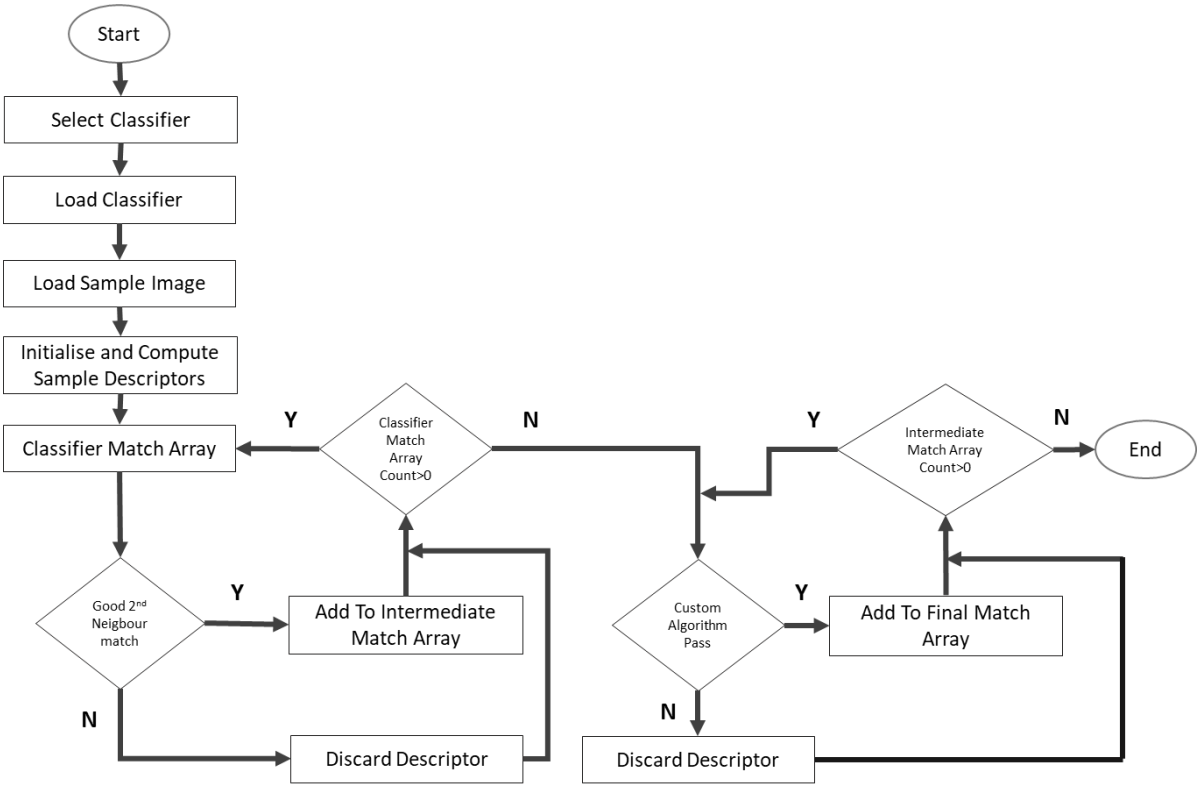


Figure 3.5. Testing procedure for the classifier-based matching.

The testing procedure for classifier-based keypoint matching is shown in Figure 3.5. The pretrained classifier is read into memory (RAM) once on system initialisation. The results section will expand on the advantages and disadvantages of doing this as well as the time-based responsiveness of this action. The use of a pretrained classifier means no manual iteration is conducted for this testing methodology. Keypoint detection and descriptor extraction is only done once for the sample test image. Classifier matching of descriptors is conducted on the entire descriptor dataset at a time as opposed to per image as done in the image-to-image comparison. The second neighbour ratio test is also employed here as an initial segregation test. A custom matching algorithm was also designed for the matching methodology and follows the following steps:

1) Compute total distance from sample descriptor for each viable descriptor found per species in viable descriptor array.
2) Compute total number of species occurrences in viable descriptor array.
3) Given total number of occurrences of a species in the viable descriptor matrix is greater than zero, compute similarity index (SI) using the Equation 3.3 where *Species Total Distance* is the sum of the distances from the sample descriptor for each species and *Species Number of Occurrences* denotes how many occurrences of that species were in the returned match array.

$$SI = \frac{Species\ Total\ Distance}{Species\ Number\ of\ Occurrences^2} \qquad \textit{Eq 3.3}$$

Equation 3.3 is at the core of the keypoint algorithm matching process and was derived taking into consideration two vital experimental observations (presented in Section 5.1):

a) The accuracy of matching, as quantified by the similarity index (SI) value, is directly proportional to the average distance. The larger the average distance, the less likely that sample is a viable match.
b) The SI value is inversely proportional to the number of occurrences of the sample in the viable match array.

Floating point string-based classifiers investigated in this thesis make use of k-NN classifiers. Binary based descriptors are best classified using Hamming distance as the measure of classification. Use of the L2-Norm to cluster binary descriptors leads to unpredictable and often incorrect results. Binary descriptors therefore substitute the k-NN algorithm with a FLANN (Fast Library for Approximate Nearest Neighbours) based clustering classifier that utilises Hamming distance [42]. A brief outline of the implementation of the classifier algorithms in the OpenCV image processing platform is given in the Section *OpenCV Classifier Implementation* that follows.

OpenCV Classifier Implementation

Experiments were conducted on two classifier implementations provided by OpenCV, one located in the *cv::ml* namespace and the other a part of the *features2d* module (located in the general *cv* namespace), which also contains implementations for the various descriptor algorithms. Classifiers saved to file utilise OpenCV`s *cv::ml* namespace. This namespace contains methods that facilitate for the creation

(*cv::ml::KNearest*) of a classifier object, training (*cv::ml::StatModel::train*) of the object with data of type *cv::ml::TrainData* and then saving the object to a text file. The *cv::ml* namespace classes and methods only accept floating point descriptors and can only brute-force cluster based on L2-Norm. They therefore cannot be used with binary descriptors which require a Hamming distance measure for accurate clustering. Classifiers that utilise the *features2d* module are RAM based. This thesis will focus on *features2d* classifiers that use an implementation of the FLANN library [42] contained in OpenCV`s *cv::FlannBasedMatcher* class [43]. These classifiers can be used with both string and binary descriptors. Binary descriptors use the *cv::flann::LshIndexParams* parameter based off work conducted by Qin et al [44]. Experimental procedure is used to determine the appropriate parameter to be used for string-based descriptors. The possible parameters chosen for experimentation are *LinearIndexParams,* which performs a linear brute force search, and *KDTreeIndexParams* which constructs a set of randomized k-d trees which are searched in parallel [43].

The k-NN algorithm is a non-parametric, lazy learning algorithm used for data classification and regression [45]. Being non-parametric, the k-nearest neighbour algorithm is ideal for data classification where no assumptions are made about the underlying data distribution [45, 46]. The k-NN algorithm utilises the L2-Norm (Euclidean distance) and is therefore used on string based float (CV_32F [47]) descriptors. Binary descriptors are best classified using Hamming distance instead. Using L2-Norm on binary descriptors may lead to incorrect clustering.

3.4.4 Moments

This experimental procedure looks at the use of moments in the qualitative and quantitative analysis of objects in images. Attention is limited to moments of monomials (Geometric moments) whose analysis will be conducted using Hu moments. The design will again be broken into training and testing phases.

Hu moments calculate seven moments (shown in Section 2.3.3) from the first ten geometric moments i.e. moments up to the third order [35–37]. Hu moments utilise a k-NN classifier and therefore follow the training procedure outlined in Section 3.4.2. The following steps outline the descriptor extraction step followed for Hu moments.

1) Derivation of image moments: Derivation of the seven Hu moments is conducted using the OpenCV function *cv::HuMoments* which runs code implementations of the equations discussed in Section 2.3.3.
2) Log transformation: Hu moments by nature have a large range in the decimal scale. The difference between H_0 and H_6 is in the order of 10^{18} in most use cases. Transformation to a base-10 log scale yields a more comparative range with smaller values, which also reduces the computational complexity of step 3 below.
3) k-NN classifier training: In the case of Hu moments, H_6 is useful in ascertaining mirror-transformation or parallel projection by analysing the sign of the value. This is an interesting characteristic however for the purposes of this thesis attention is focused more on the properties elucidated by analysis of the magnitude more than the sign of the magnitude. The absolute value of H_6 is therefore used in training the Hu moment classifier and no further analysis is carried out on its relative sign.

The testing procedure for classifier-based Hu moments matching is shown in Figure 3.6.

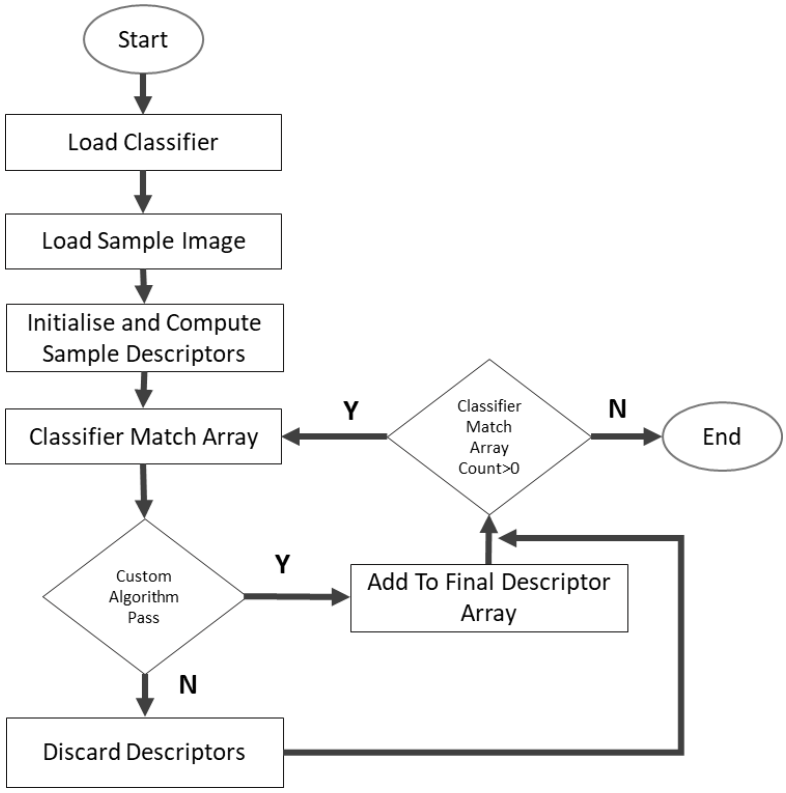


Figure 3.6. Hu moments testing procedure process flow.

The k-NN classifier is used to find the nearest neighbours of a given set of moments. These nearest neighbours are then used as the input to the moments custom matching algorithm. The moments custom matching algorithm used for moments analysis is the same as that used by the keypoint matching algorithm discussed in Section 3.4.3. Experimental procedure will show the qualitative and quantitative effectiveness of using this algorithm with the moment's classifier.

3.4.5 Hashing

Several hashing algorithms are available as discussed in the literature review. Focus is limited to four of these for experimentation, namely:

- Average hash (dHash)
- pHash
- Colour Moment hash
- Block Mean hash

Radial variance hash and Marr Hildreth hash were excluded from experimental procedure as they have little or no resiliency to scaling, rotation or salt and pepper noise [29 p. 77, 48]. Several open source and commercial libraries are available for use in image hashing, an example being the open source pHash library from phash.org [49]. Experimental procedure will utilise the hashing library provided by the OpenCV image processing library`s *cv::img_hash* namespace [50], which contains image hashing functions for the four algorithms under consideration.

Procedure is again broken down into training and testing phases. Training also follows the process outlined in Section 3.4.2 and involves the recursive extraction of hash values from test images and training a k-NN classifier. Image hash values are relatively small and extraction of the hash values is not a computationally intensive process, even for the DCT/DWT based pHash algorithms. Hash comparison makes use of the Hamming distance to ascertain similarity, except for the Colour Moment hash, which takes floating point inputs and returns a double value. The testing procedure for hash algorithms is the same as that outlined for Hu moments in Figure 3.6.

4 Experimental Results

This chapter presents results obtained from the various experimental procedures and methods outlined in previous chapters. Results are presented for the three classification techniques under investigation, namely, keypoint matching, hashing and moments. Each of these classification techniques will use the three metrics indicated below to measure their qualitative and quantitative effectiveness.

1) **Computation speed**: FLORA is designed for real-time processing therefore underlying processes and computational tasks must execute efficiently and timeously to afford the overall system real-time capability. Speed is therefore an important metric of the system.
2) **Matching accuracy**: Matching accuracy is a measure of the system`s effectiveness in providing a correct match for a given input sample.
3) **Similarity Index**: Unlike the matching accuracy, the similarity index (SI) seeks to measure the reliability of the system by ascertaining a confidence measure of a found match in relation to second, third and N-th matches. The similarity index is therefore used to provide a qualitative analysis of the system.

4.1 Keypoint Matching Algorithm Analysis

The keypoint matching testing procedure builds on the work conducted by several authors who have published works with quantitative and qualitative results obtained from comparison of a base image with its transposed or transformed variant [14, 18, 19, 22]. Unlike most of these published works, the FLORA project is unique in that it suggests an image matching technique that provides a similarity index for similar objects in images i.e. FLORA attempts to match objects that are structurally similar but with naturally occurring variations. After presentation of experimental results, a summary of the keypoint matching analysis is given in Section 4.1.3.

4.1.1 Image-to-Image Comparison

Results obtained from direct image-to-image analysis are used as a base for subsequent tests. Image-to-image comparison is not optimised and involves the recursive loading of training images for comparison with a sample image. Immutable system parameters which are constant regardless of the training or testing methodology used were also tested under image-to-image comparison. The scope of this test is therefore composed of the following analyses:

- Effect of image scaling and rotation on matching and system accuracy. These are the affine transformations anticipated to most likely be the commonly encountered by the FLORA system.
- Keypoint extraction, descriptor computation and descriptor comparison execution speed for the various keypoint matching algorithms.
- Effectiveness of the second neighbour ratio test and image-to-image custom matching algorithm in match segregation.

Analysis is conducted on the leaves and stems databases using a random sample of 5 leaf and 6 stem images from the respective databases.

Effect of Scaling on Matching

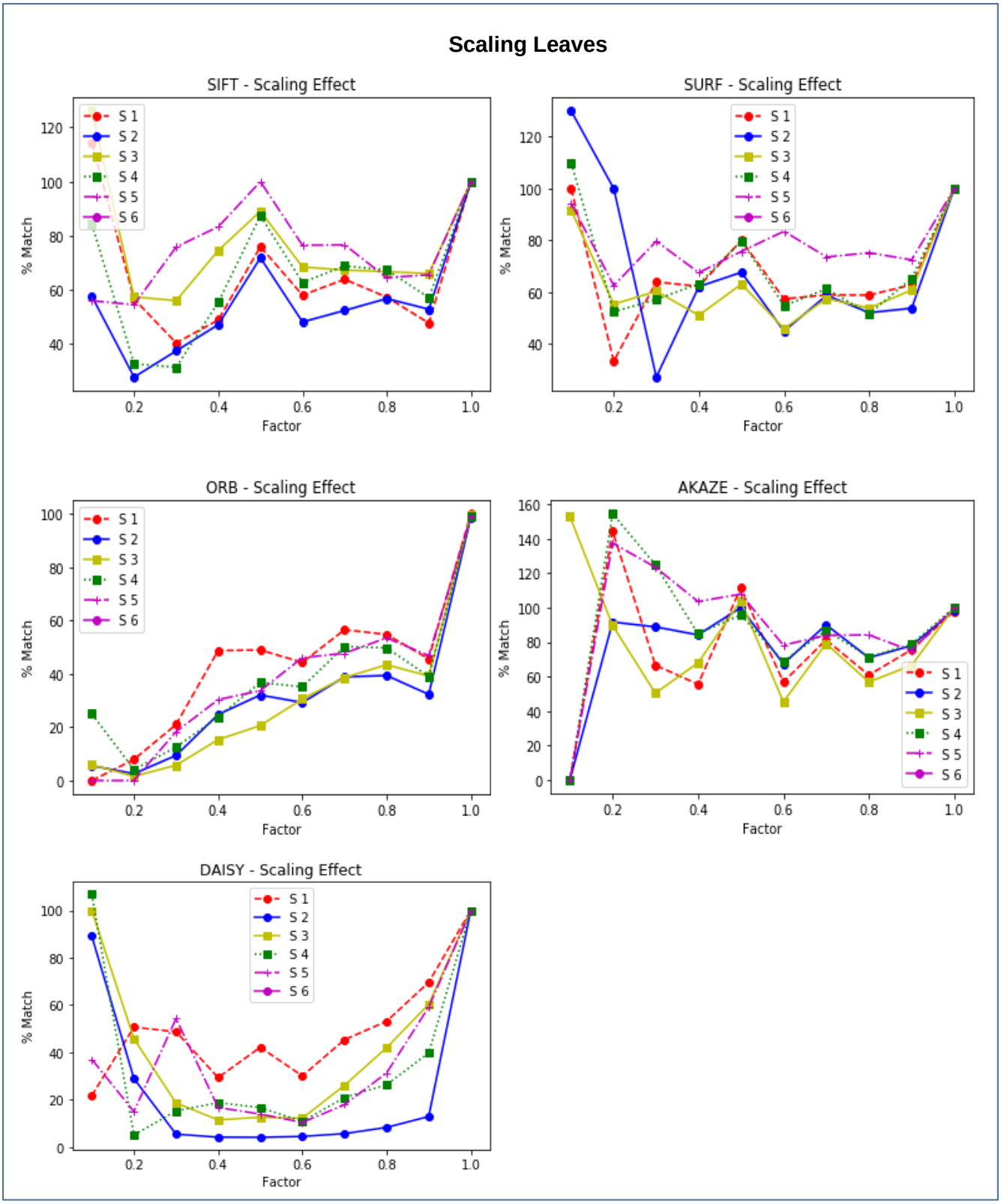


Figure 4.1. Analysis on effect of scaling on leaf samples.

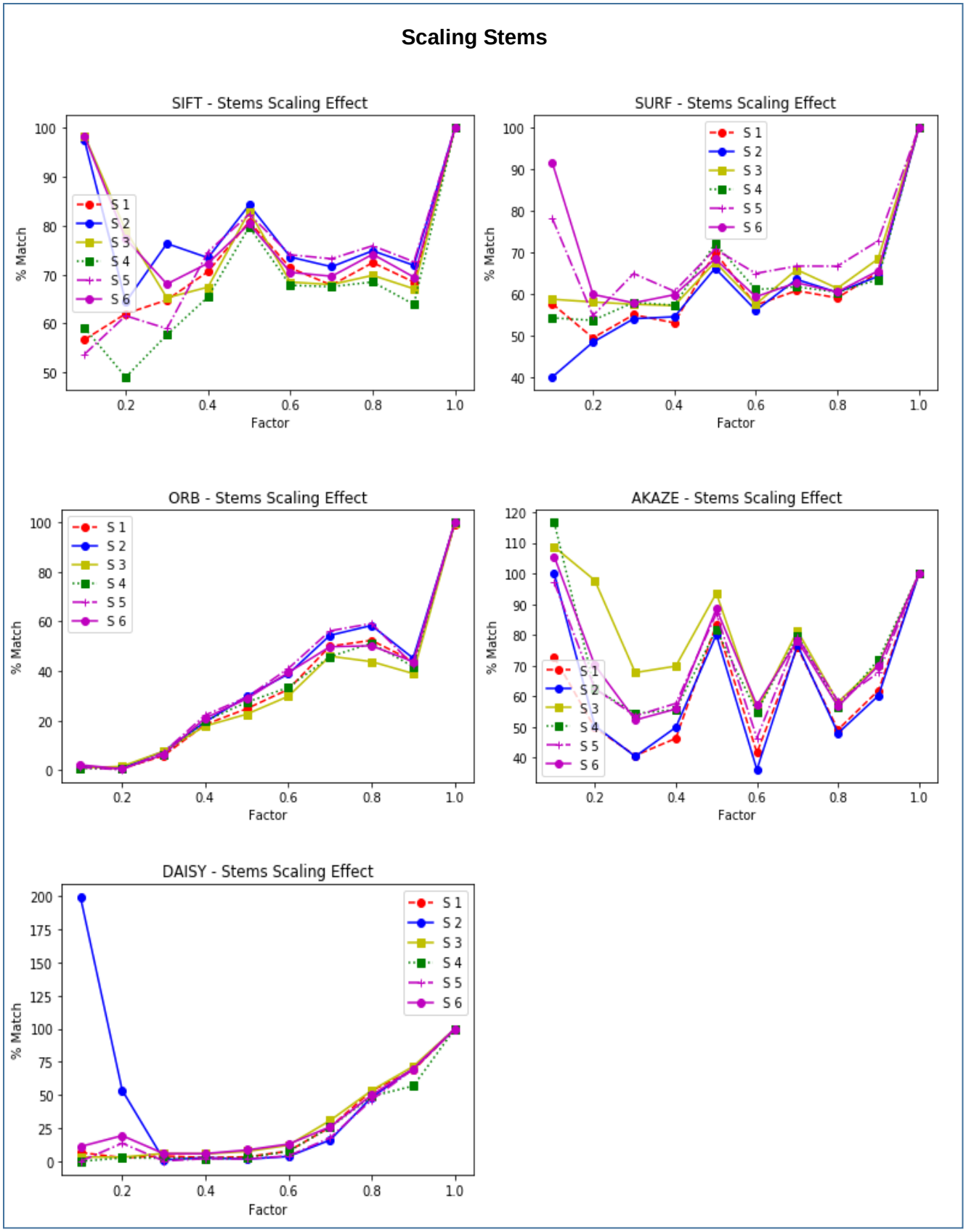


Figure 4.2. Analysis on the effect of scaling on stem samples.

Scaling results presented in Figures 4.1 and 4.2 show that string descriptor-based algorithms have the greatest stability, with SIFT and SURF having a matching accuracy of approximately 65% and a variance of ±10% across scaling factors. The best accuracy is obtained when the scaling factor is greater than 0.4 for SIFT and SURF descriptors. ORB has a more linear response to scaling but even the best matching accuracy obtained for all scaling factors is still less than that obtained for string descriptors. AKAZE also has a good scaling average response with an accuracy of approximately 65%. AKAZE however shows extreme variation across scaling factors, with variations of up to 45% in matching accuracy.

Effect of Rotation on Matching

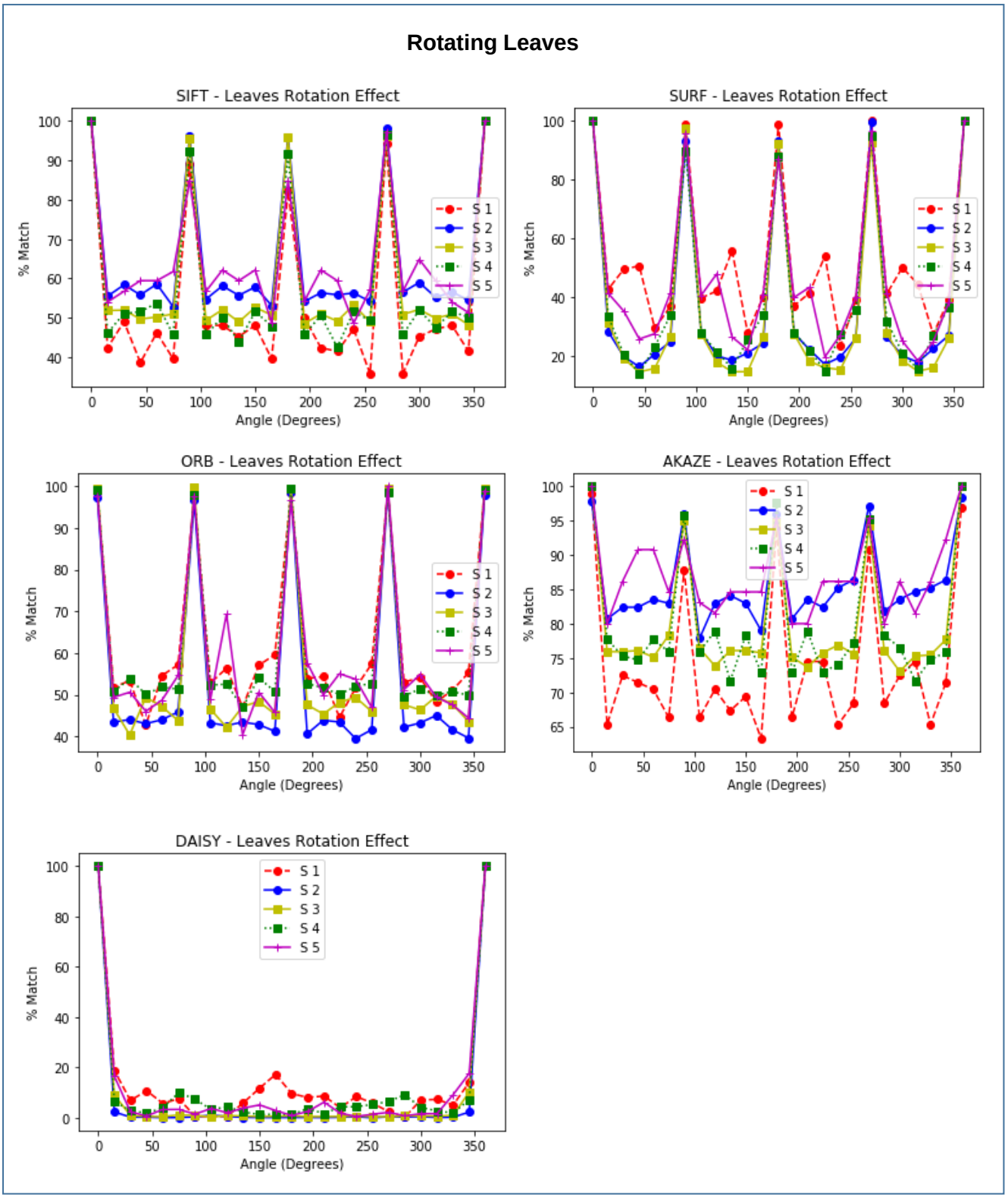


Figure 4.3. Analysis on effect of rotation on leaf samples.

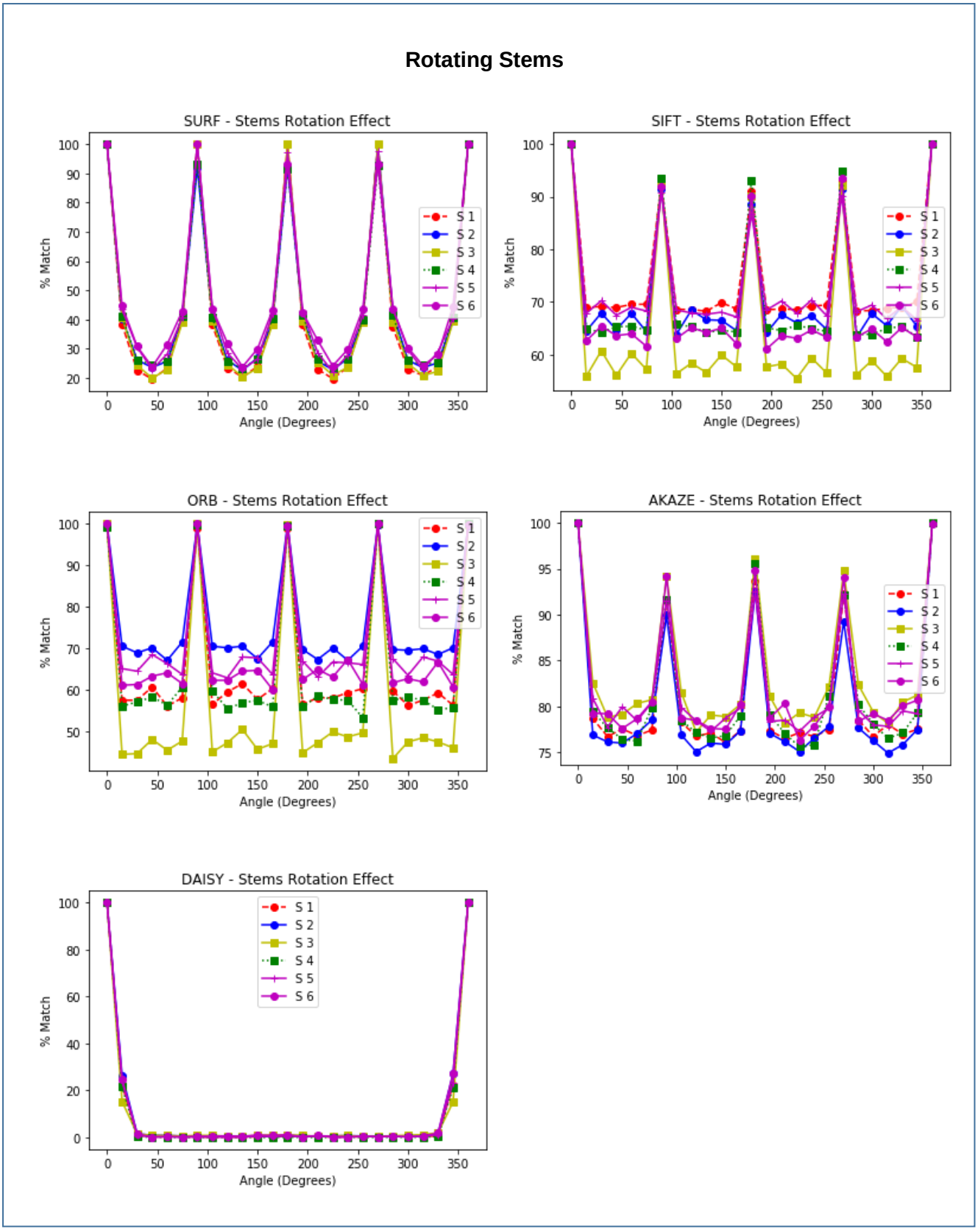


Figure 4.4. Analysis on the effect of rotation on stem samples.

Leaf and stem rotation analysis presented in Figures 4.3 and 4.4 showed that SIFT, ORB and AKAZE have the greatest resiliency to rotation, having a matching accuracy ≥65% from 0 to 2π radian angles of rotation. Daisy has the worst rotation invariance performance compared to other algorithms. It was primarily designed for wide baseline stereo matching and exhibits the same robustness to contrast, scaling and brightness as histogram based descriptors [15, p. 13] however analysis has shown it to be susceptible to rotation transformations.

Computational Speed Analysis

This test was designed to run a quantitative speed comparison analysis for the various keypoint matching algorithms for the different stages of keypoint matching, namely, keypoint extraction, descriptor computation and descriptor matching. This analysis is built on the work conducted by S. Tareen and Z. Saleem in [16] and E. Karami et al in [14].

Table 4.1. Computation times for the various keypoint algorithms.

Descriptor	Key points 1	Key points 2	% Match	Keypoints 1 Calc Time	Keypoints 2 Calc Time	Match Time (ms)	2nd Neighbour Calc Time	Total (ms)
Leaf Sample 1								
SIFT	106	106	100	693.153	581.395	42.1873	0.5715	1317.31
SURF	201	201	100	1424.33	382.435	66.0377	0.5041	1873.31
ORB	323	323	97.8328	664.064	671.312	354.955	0.8744	1691.2
ORB1000	325	325	99.0769	652.832	680.478	359.772	1.2923	1694.37
ORB10000	325	325	98.7692	716.281	719.473	346.394	1.1981	1783.35
AKAZE	98	98	98.9796	1360.97	1336.82	107.457	0.4795	2805.73
DAISY	450	450	100	1800.71	1801.76	199.353	1.3154	3803.13
Leaf Sample 2								
SIFT	106	1495	5.66038	556.803	1162.35	134.135	0.1597	1853.45
SURF	201	1155	1.49254	368.184	831.52	98.7466	0.2364	1298.69
ORB	323	497	0.928793	656.048	1252.2	347.533	0.5348	2256.32
ORB1000	325	931	0.615385	670.672	1270.29	413.675	0.5636	2355.2
ORB10000	325	6428	0.615385	648.35	1413.42	1109.87	0.3887	3172.02
AKAZE	98	176	2.04082	1383.66	2630.37	119.819	0.1232	4133.97
DAISY	450	5405	3.55556	1894.37	3726.3	591.608	1.0965	6213.38
Leaf Sample 3								
SIFT	106	1933	1.88679	575.405	1678.29	142.662	0.2498	2396.61
SURF	201	1129	9.95025	369.732	1375.39	111.692	0.3023	1857.11
ORB	323	500	0	650.838	1784.84	337.839	0.3476	2773.86
ORB1000	325	1000	0	661.39	1800.84	412.659	0.4715	2875.36
ORB10000	325	5656	0	651.724	1920.24	968.51	0.386	3540.86
AKAZE	98	627	0	1367.1	3650.38	174.367	0.1107	5191.96
DAISY	450	3859	2	1793.04	5444.67	478.2	0.5697	7716.48
Leaf Sample 4								

SIFT	106	223	3.77358	553.097	945.378	43.0554	0.1392	1541.67
SURF	201	383	1.49254	381.486	624.947	80.7066	0.2625	1087.4
ORB	323	451	0	654.339	1084.58	356.343	0.5003	2095.76
ORB1000	325	745	0	664.07	1093.92	397.786	0.4272	2156.21
ORB10000	325	2632	0	660.599	1156.34	607.617	0.3758	2424.93
AKAZE	98	166	0	1339.69	2279.5	114.203	0.1173	3733.51
DAISY	450	1651	1.11111	1788.9	3073.12	306.163	0.5509	5168.73
Leaf Sample 5								
SIFT	106	37	18.9189	567.385	1044.48	36.9475	0.2033	1649.02
SURF	201	148	0.675676	385.401	667.369	63.9562	0.2886	1117.01
ORB	323	89	0	694.138	1206.77	280.618	0.6231	2182.15
ORB1000	325	89	2.24719	663.974	1185.13	281.116	0.4252	2130.65
ORB10000	325	89	1.1236	661.06	1263.42	302.249	0.4754	2227.21
AKAZE	98	65	0	1359.05	2767.45	143.615	0.1662	4270.28
DAISY	450	544	2.22222	1930.94	3562.71	197.522	0.6386	5691.81
Stem Sample 1								
SIFT	4517	4517	100	2785.83	2649.23	1989.22	15.6653	7439.94
SURF	5667	5667	100	4394.47	3407.41	2407.26	21.986	10231.1
ORB	500	500	99.8	2875.6	2734.47	532.832	1.4275	6144.33
ORB1000	1000	1000	100	2675.63	2663.71	1048.41	2.7552	6390.5
ORB10000	10000	10000	100	2955.05	2976.15	16977.3	30.3903	22938.9
AKAZE	5357	5357	100	5461.52	5557.76	8558.21	15.7353	19593.2
DAISY	8080	8080	100	8005.33	8055.66	3909.47	23.9784	19994.4
Stem Sample 2								
SIFT	4517	3085	0.226904	2679.94	1715.62	1809.79	6.5524	6211.91
SURF	5667	4068	0.368732	3484.9	2810.88	2141.78	7.4438	8445
ORB	500	500	1.4	2862.69	1991.74	556.446	0.7496	5411.63
ORB1000	1000	1000	0.4	2967.17	1796.99	1091.87	1.186	5857.21
ORB10000	10000	10000	0.08	3155.85	2073	16120.1	14.5146	21363.5
AKAZE	5357	4018	0.099552	5805.02	3921.29	8026.78	7.1417	17760.2
DAISY	8080	6430	0.0933126	7919.86	5270.82	3690.59	10.4628	16891.7
Stem Sample 3								
SIFT	4517	2008	0.448207	2971.05	2099.31	1714.47	6.9257	6791.75
SURF	5667	1643	1.46074	3438.43	1947.99	1968.9	7.2043	7362.53
ORB	500	500	0.2	2807.54	2189.5	553.16	0.6421	5550.84
ORB1000	1000	1000	0.3	2831.85	2168.77	1180.68	1.466	6182.77
ORB10000	10000	5112	0.176056	3008.91	2261.7	13039	12.6411	18322.2
AKAZE	5357	1194	1.00503	5748.33	4268.35	6019.16	7.2096	16043
DAISY	8080	4195	0.405244	7916.49	6278.03	3308.47	11.5192	17514.5
Stem Sample 4								
SIFT	4517	4075	0.0490798	2814.58	2225.54	2057.01	7.306	7104.43
SURF	5667	3689	0.460829	3748.76	2046.19	2242.78	10.2313	8047.96
ORB	500	500	0.6	2894.5	2080.15	591.955	1.9686	5568.57
ORB1000	1000	1000	0.4	3069.75	1902.92	1160.83	1.4522	6134.95
ORB10000	10000	9836	0.0305002	3439.72	2477.45	18050.7	13.6066	23981.5
AKAZE	5357	1683	0.534759	6131.92	3758.57	6998.7	8.9157	16898.1
DAISY	8080	7919	0.151534	8343.33	5623.77	3839.43	12.3989	17818.9
Stem Sample 5								
SIFT	4517	1107	1.17435	2740.97	1788.24	1580.34	6.6272	6116.18
SURF	5667	1621	1.11043	3641.42	1758.72	2208.86	9.5287	7618.54
ORB	500	500	1.4	3048.58	2183.22	601.982	1.5306	5835.32

ORB1000	1000	1000	0.5	3032.63	2139.67	1097.57	1.3402	6271.21
ORB10000	10000	5061	0.138313	3260.35	2202.13	14611.8	21.8517	20096.2
AKAZE	5357	1308	0.382263	6387.03	5131.37	6452.37	7.7377	17978.5
DAISY	8080	3407	0.352216	8082.09	6880.69	3713.42	13.879	18690.1
Stem Sample 6								
SIFT	4517	1382	0.868307	3357.6	1344.13	1939.96	6.8691	6648.56
SURF	5667	1389	2.73578	3741.91	1212.27	2130.38	8.6936	7093.25
ORB	500	500	0.6	3281.92	1548.12	604.941	0.6064	5435.59
ORB1000	1000	1000	0	3355.86	1378.58	1160.66	2.2855	5897.39
ORB10000	10000	5621	0.0711617	3485.64	1505.62	15110.5	14.5888	20116.4
AKAZE	5357	1134	0.617284	7009.43	2964.44	6999.14	8.1818	16981.2
DAISY	8080	2615	0.458891	9209.35	4427.25	4200.42	13.0435	17850.1
Mean Values								
SIFT	2512	1815	21.18	1845.07	1566.72	1044.53	4.66	4460.98
SURF	3182	1917	19.98	2307.18	1551.37	1229.19	6.06	5093.81
ORB	419	441	18.43	1917.30	1702.45	465.33	0.89	4085.96
ORB1000	693	826	18.50	1931.44	1643.75	782.28	1.24	4358.71
ORB10000	5602	5523	18.27	2058.50	1815.36	8840.37	10.04	12724.28
AKAZE	2966	1438	18.51	3941.25	3478.75	3973.98	5.08	11399.06
DAISY	4611	4050	19.12	5334.95	4922.25	2221.33	8.13	12486.66

Table 4.1 shows results obtained from computational time analysis on the various algorithms under investigation. The worst mean computational times are highlighted in red in the table. SIFT and SURF both have comparable computation times for leaf samples. On average, for image-to-image analysis, they both outperformed the other algorithms for leaf samples which have low keypoint bin size. SIFT and standard ORB (ORB500) however gave the best results with high keypoint bins. Daisy and AKAZE show the weakest performance overall, with Daisy and ORB10000 being the most computationally intense algorithms. Daisy is a dense descriptor and computes descriptors for every pixel in an image or input patch. As expected with the ORB descriptor, the larger the value of N used to return the number of quality matches the more expensive the computation [14 p. 2, 16 p. 3].

Second Neighbour Ratio Test and Image-to-Image Custom Matching Algorithm Analysis

The effectiveness of the second neighbour ratio test was analysed by D. Lowe in [18, p. 20]. In his analysis, D. Lowe showed that for a match to be considered viable, its distance to the second nearest match had to be significantly larger. The ratio between the closest and second closest match can therefore be used as an estimate of false matches in a feature set as well offering insight into feature ambiguity [18, p. 20]. The

test conducted in this section ascertained the best ratio for the fynbos leaf and stem datasets.

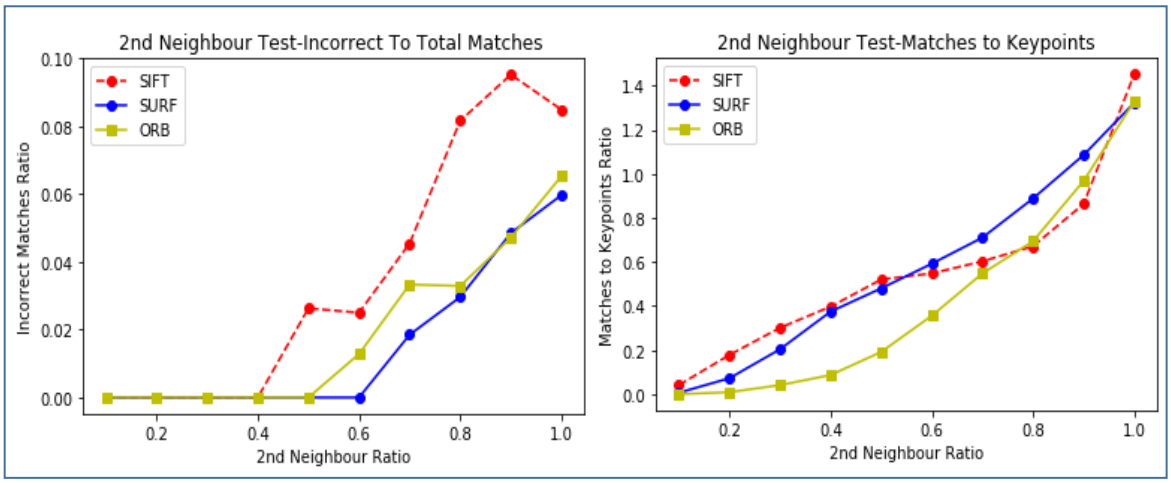


Figure 4.5. Graphs showing experimentation on an image with its scaled and resolution adjusted version used to ascertain the effect of the 2nd neighbour test on: Left) Accuracy of matches returned and Right) The matches to extracted keypoints ratio.

Figure 4.5 shows results obtained from running the second neighbour ratio test on an image and its scaled and resolution adjusted version. Comparing a sample image with its scaled and resolution adjusted version in this test ensured some variability was introduced while still maintaining a high degree of similarity between the test images. This creates a controlled test case in which results can be attributed mainly to the second neighbour parameters. Experimentation using the SIFT, SURF and ORB keypoint algorithms showed results consistent with D. Lowes analysis in [18, p. 20]. A sharp rise in incorrect matches is observed beyond a second neighbour ratio of 0.6 with the fynbos dataset (Figure 4.5, left image). Matches to keypoints analysis (Figure 4.5, right image) also shows an average match to keypoints ratio greater than 0.5 beyond a second neighbour ratio of 0.6. The matches to keypoints analysis also showed unreliable keypoint matching beyond a second neighbour ratio of 0.9, with the matches to lowest keypoint size ratio going beyond unity, raising the probability of keypoint feature ambiguity. From the graphs, a factor of 0.6 minimises the incorrect matches ratio and maximises the matches to keypoints ratio for the fynbos dataset.

The image-to-image keypoint custom match algorithm makes use of two metrics which form the basis of the underlying tests.

i) Image-to-image comparison keypoint percentage match.
ii) Keypoint variance and its usefulness in match segregation.

Keypoint matching is a complex process that is sensitive to factors such as rotation, scaling, contrast and noise. For most cases, percentage match rates of greater than 80% can be achieved when comparing an object with its affine transformed or transposed copy. The FLORA project however relies on matching of geometrically and structurally similar but different objects, adding another dimensionality of complexity. Using the Proteaceae Leucospermum reflexum var. luteum (Mountain Fynbos - also identified as Sample 22 in Table 3.1) as the stem control sample and a Protea Lanceolata (Lance leaf sugarbush- also identified as Sample 19 in Table 3.1) leaf as the leaf control sample, per sample percentage matches were calculated for the combined leaf and stem datasets.

Table 4.2. Top 10 results obtained from running keypoint algorithms across the entire dataset on a mountain fynbos stem sample (Sample 22).

	Stems - Keypoint Algorithm														
	SIFT			SURF			ORB			ORB1000			AKAZE		
	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID
1	50	98.7138	17	5.48	93.9918	6	1.8	0	8	1.51	73.5	6	19.23	96.6365	19
2	50	99.3569	19	4.62	78.6008	17	1.8	0	8	1.5	80	19	8.33	95.3428	6
3	40	99.1961	19	3.78	78.2716	17	1.8	0	8	1.2	0	8	8.33	93.7904	17
4	25	98.7138	17	3.65	88.7243	17	1.8	0	22	1	0	24	7.41	96.5071	6
5	22.22	98.5531	6	3.54	83.7037	19	1.66	52	6	0.93	3	13	6.67	96.119	6
6	20	98.3923	17	3.39	85.4321	19	1.6	0	9	0.9	0	8	6.67	96.119	19
7	18.18	98.2315	19	3.319	80.1646	17	1.6	0	13	0.8	0	9	6.56	92.1087	6
8	16.67	98.0707	17	2.96	86.0905	19	1.6	0	17	0.8	0	17	5.88	95.6016	17
9	15.38	95.8199	23	2.95	80.4938	17	1.6	0	22	0.8	0	17	5.56	95.3428	19
10	15	96.7846	23	2.89	85.7613	19	1.6	0	24	0.8	0	17	5	92.238	19

Table 4.2 highlights percentage match accuracies obtained for stem image-to-image analysis using SIFT, SURF, ORB, ORB1000 and AKAZE algorithms. Green rows

highlight correct matches. ORB500 gave two positive matches while the remaining algorithms had no positive matches.

Table 4.3. Leaf keypoint analysis on a Lance leaf sugarbush sample.

	Leaves - Keypoint Algorithm														
	SIFT			SURF			ORB			ORB1000			AKAZE		
	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID
1	**481.8**	98.51	16	**21.76**	69.75	6	**1.76**	40.88	17	**2.20**	77.3	11	**8**	86.1878	2
2	**409.1**	98.47	22	**10.58**	60.64	6	**1.76**	27.47	17	**2.20**	77.3	11	**6**	83.9744	2
3	**372.7**	97.05	17	**8.82**	34.11	6	**1.32**	54.6	3	**1.76**	77.3	2	**6**	91.2281	3
4	**363.6**	97.71	22	**8.24**	37.73	6	**1.32**	54.6	5	**1.76**	74.72	10	**6**	94.1793	11
5	**327.27**	99.02	18	**7.06**	34.61	6	**1.32**	54.6	8	**1.76**	77.3	17	**6**	49.4949	17
6	**327.27**	98.41	22	**6.47**	68.69	6	**1.32**	54.6	11	**1.76**	46.58	17	**6**	59.3496	23
7	**318.18**	98.11	22	**6.47**	33.33	17	**1.32**	40.26	17	**1.76**	27.47	17	**6**	72.3757	23
8	**300**	97.59	17	**5.88**	11.91	17	**1.32**	54.6	17	**1.76**	77.3	21	**4**	86.1496	2
9	**290.91**	98.77	18	**5.29**	46.71	6	**1.32**	42.38	18	**1.76**	77.3	23	**4**	87.3737	5
10	**281.82**	97.36	2	**4.71**	44.44	6	**1.32**	54.6	21	**1.32**	77.3	2	**4**	81.0606	5

Table 4.3 shows leaf keypoint analysis on the Lance leaf sugarbush sample. The leaf image-to-image comparison gave more positive results (in green) with the keypoint algorithms than the stem analysis. This can be attributed to the more regular structure of leaves. There is less variability in leaf structure than in stems resulting in more stable match results even without keypoint segregation. The results also show a higher collision rate with leaves resulting in percentage match rates much higher than 100%. This again can be attributed to the much lower number of keypoints extracted from leaves with a low level of variability.

The keypoint algorithms under investigation use either corners or blobs in keypoint extraction. Experimental investigation showed that on average, stems produce a much larger number of keypoints compared to leaf samples. Figures 4.6 and 4.7 give a visual representation of this observation.

Figure 4.6. SIFT blob-based extraction. Left. Mimetes Cucullatus (Common Pagoda) Stem sample with 4517 extracted keypoint. Right. Leucospermum Grandiflorum (grey leaf fountain pincushion) leaf sample with 223 extracted keypoints.

Figure 4.7. ORB10000 corner-based extraction. Left: 10000 (limit) keypoints extracted from the Mimetes Cucullatus (Common Pagoda) stem sample. Right: 2632 keypoints extracted from the grey leaf fountain pincushion sample.

Keypoint variance is a good segregation technique in system`s were image resolution and aspect ratio do not vary widely. Samples from geometrically similar objects produce the same relative number of keypoints. Tables 4.4 and 4.5 below are a result of applying keypoint segregation to the results obtained in Table 4.2 and 4.3.

Table 4.4. Result of using keypoint variance as a segregation parameter using stems.

	Stems - Keypoint Algorithm														
	SIFT			SURF			ORB			ORB1000			AKAZE		
	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID
1	**4.50**	13.85	22	**1.16**	27.89	22	**1.8**	0	8	**1.2**	0	8	**1.26**	28.2018	8
2	**4.29**	6.43	22				**1.8**	0	8	**1**	0	24	**1.09**	5.17464	18
3	**3.65**	25.24	8				**1.8**	0	8						
4	**3.63**	24.76	22				**1.8**	0	22						
5	**3.33**	22.83	22				**1.6**	0	9						
6	**3.06**	16.08	22				**1.6**	0	13						
7	**3.05**	10.37	22				**1.6**	0	17						
8	**2.90**	22.50	22				**1.6**	0	22						
9	**2.89**	11.06	22				**1.6**	0	24						
10	**2.89**	15.94	22				**1.4**	0	8						

A ±30% keypoint variance is used in Table 4.4 as a segregation parameter. Significant improvement is observed in the number of correct matches obtained. When matching stem samples, blob-based string descriptor algorithms show the best matching capabilities with SIFT correctly matching the mountain fynbos stem sample nine out of ten times (matches highlighted in green).

A ±30% variance threshold is also used in Table 4.5 on the Lance Leaf Sugarbush leaf sample. An increase in quality, as opposed to quantity, of the matched results is observed (correct matches highlighted in green). Three of the algorithms correctly place the sample at position one. Binary, corner-based descriptors show the most precision with keypoint segregation when matching leaves in this analysis.

Table 4.5. Result of using keypoint variance as a segregation parameter using leaves.

	Leaves - Keypoint Algorithm														
	SIFT			SURF			ORB			ORB1000			AKAZE		
	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID	% Match	% Variance	Sample ID
1	**81.81**	0	6	**5.88**	11.91	17	**1.76**	27.47	17	**1.761**	27.47	17	**4**	18.03	6
2	**77.77**	18.18	6	**2.94**	18.26	6							**4**	19.35	17
3	**55.55**	18.18	6	**2.94**	8.11	6							**2.56**	22	17
4	**37.5**	27.27	17	**2.35**	9.09	15							**2.38**	16	17
5	**37.5**	27.27	17	**2.35**	20.56	15							**2.22**	10	6
6	**37.5**	27.27	17	**2.35**	0.58	17							**2.08**	4	17
7	**36.36**	15.38	6	**2.35**	27.96	23							**2.04**	2	19
8	**36.36**	15.38	6	**2.01**	12.35	17							**2**	27.53	6
9	**36.36**	21.42	15	**1.96**	10	19							**2**	0	6
10	**36.36**	26.67	17	**1.91**	7.64	19							**2**	7.41	6

4.1.2 Classifier Based Comparison

As indicated in Section 3.4.2, this comparison focuses on two types of classification algorithms namely those located in OpenCV`s *cv::ml* namespace and those located in the features2d class. The keypoint classifier testing procedure focuses on the following:

- Percentage match accuracy of the various classifiers using the custom matching algorithm.
- Speed of sample matching.
- Keypoint algorithm effectiveness in inter species and inter family comparison.
- An analysis on algorithm effectiveness when matching geometrically similar and dissimilar structures. This is important in formulating an effective SI index. A robust matching algorithm must produce wide SI index variability between dissimilar images.

Data Splitting

Data splitting follows the k-fold procedure described in Section 3.4.1 Table 4.6 shows the process of using a 4-fold strategy on the dataset and the results obtained.

Table 4.6. Iterative training and testing over the four folds showed split 3 gave the best accuracy for keypoint matching using ORB500 keypoint descriptors.

					Avg Accuracy
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	42.50%
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	51.30%
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	56%
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	36%

Training Data Testing Data

Cv::ml Namespace File Classifiers

Basing on results obtained in 4.1.1 and the methodology discussed in 3.4.3, this section shows experimental results obtained from running test images through a file brute-force k-NN classifier using the SURF and SIFT blob-based string descriptors. As discussed in 3.4.3, binary descriptors cannot be used with OpenCV`s *cv::ml* namespace as it only accepts floating point image descriptors for analysis. The output of this experimental procedure is the calculation of the similarity index (SI) which is then used to classify leaf and stem samples.

Table 4.7. A snippet of 30 of the 103 test results obtained from running test images through a SURF file k-NN classifier.

			Correct Match											
Sample ID	**Calc Time (ms)**	**Matches**	**No. 1**	**Top 5**	**Top 10**	**Match 1**	**Match 1 Occurrences**	**Match 1 SI**	**Match 2**	**Match 2 Occurrences**	**Match 2 SI**	**Match 3**	**Match 3 Occurrences**	**Match 3 SI**
1	74518.5	15	FALSE	FALSE	FALSE	4	1	0.000512	18	3	0.001322	10	2	0.00235
1	68050.0	8	FALSE	TRUE	TRUE	9	3	0.001292	16	3	0.001506	17	1	0.01159
1	54717.3	5	FALSE	FALSE	FALSE	15	1	0.008559	10	1	0.009655	2	1	0.01769
2	52720.3	10	TRUE	TRUE	TRUE	2	5	0.000158	16	1	0.008015	7	1	0.01287
2	40492.5	7	TRUE	TRUE	TRUE	2	3	0.000611	5	1	0.002240	22	1	0.00523
2	39534.2	7	FALSE	TRUE	TRUE	15	1	0.001896	2	1	0.002171	5	2	0.00241
3	60770.9	8	FALSE	TRUE	TRUE	2	1	0.003975	22	1	0.008274	23	1	0.01227
3	35258.6	6	TRUE	TRUE	TRUE	3	4	0.001286	10	2	0.005057	1	0	2.15E+09
4	82963.2	19	TRUE	TRUE	TRUE	4	5	0.000898	18	3	0.001708	9	2	0.00232
4	70955.7	7	TRUE	TRUE	TRUE	4	4	0.001173	7	1	0.007696	20	1	0.02709
4	61879.0	9	FALSE	FALSE	FALSE	17	2	0.003143	13	2	0.004122	20	1	0.00426
4	171624	18	TRUE	TRUE	TRUE	4	4	0.000241	9	1	0.000789	16	3	0.00126
4	65472.6	9	FALSE	TRUE	TRUE	11	1	0.006380	16	1	0.007693	23	1	0.01256
5	70625.1	6	FALSE	TRUE	TRUE	2	1	0.005125	21	2	0.007805	20	1	0.01030
5	62897.7	11	TRUE	TRUE	TRUE	5	3	0.000859	7	2	0.003040	24	2	0.00573
5	103037	169	TRUE	TRUE	TRUE	5	163	6.88E-07	11	1	0.005851	23	1	0.00695
5	87492.5	6	FALSE	TRUE	TRUE	13	1	0.002538	5	2	0.006981	8	1	0.01018
5	53724.1	14	TRUE	TRUE	TRUE	5	8	0.000124	8	1	0.002294	6	1	0.00294
6	75941.1	42	TRUE	TRUE	TRUE	6	40	6.34E-07	19	1	0.000706	23	1	0.00248
6	40914.2	8	TRUE	TRUE	TRUE	6	6	0.000220	12	1	0.000430	15	1	0.00728
6	24079.9	9	TRUE	TRUE	TRUE	6	7	4.45E-05	19	1	0.003345	5	1	0.00828
6	51069.8	13	TRUE	TRUE	TRUE	6	11	8.41E-06	7	1	0.000298	17	1	0.00067
7	84608.0	8	TRUE	TRUE	TRUE	7	2	0.002961	13	2	0.004857	16	1	0.01263
7	53176.4	14	FALSE	TRUE	TRUE	23	1	0.001139	7	3	0.001223	5	2	0.00159
8	77175.3	5	TRUE	TRUE	TRUE	8	3	0.000770	24	1	0.011264	22	1	0.01655
8	88342.2	17	TRUE	TRUE	TRUE	8	3	0.000867	18	3	0.001948	7	2	0.00274
8	58929.3	23	TRUE	TRUE	TRUE	8	9	4.56E-05	22	2	0.000227	7	2	0.00053
9	86172.5	12	FALSE	TRUE	TRUE	13	2	0.003053	7	3	0.003596	8	2	0.00559
9	64845.5	12	FALSE	FALSE	FALSE	17	3	0.002799	18	2	0.003990	8	2	0.00463
9	68176.6	11	FALSE	TRUE	TRUE	8	4	0.001142	9	3	0.001170	17	2	0.00374

Table 4.7 shows test results obtained from running test images through a SURF file k-NN classifier then extracting the similarity index in the custom matching algorithm. The SURF k-NN classifier initial load time was 250s. For the unified dataset (leaves and stems), average calculation time per sample was 66.6s. The classifier correctly matched 60.4% of the unified dataset test samples i.e. the classifier found the correct match at match number one. 82.35% of the samples were correctly classified in the top 5 matches and 89.22% of the samples were correctly classified in the top 10 matches. For the stem dataset, 70.3% of stem samples correctly matched at number

one, 87.5% test samples were correctly classified in the top 5 and 92.2% were correctly classified in the top 10. For the leaf dataset, 47.1% of leaf samples correctly matched at number one, 79.4% were correctly classified in the top 5 and 88.2% were correctly classified in the top 10.

Table 4.8 shows test results obtained from running test images through a SIFT k-NN classifier then extracting the similarity index in the custom matching algorithm. The SIFT k-NN classifier load time was 261.9s. For the unified dataset (leaves and stems), average calculation time per sample was 43.7s. The classifier correctly matched 52% of the test samples i.e. the classifier found the correct match at match number one. 74% of the samples were correctly classified in the top 5 as well the top 10 matches. For the stem dataset, 56.7% of stem samples correctly matched at number one, 77.6% test samples were correctly classified in the top 5 and the same value was true for test samples classified in the top 10. For the leaf dataset, 41.2% of leaf samples correctly matched at number one, 61.7% were correctly classified in the top 5 as well as in the top 10.

Table 4.8. A snippet of 30 of the 103 test results obtained from running test images through a SIFT k-NN classifier.

SIFT Load Time: 261.9s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	58627.4	5	TRUE	TRUE	TRUE	1	3	1751.4	11	1	9366.0	9	1	16068.0
1	55756.2	8	FALSE	FALSE	FALSE	16	2	3307.8	11	2	5483.5	7	1	18870.0
1	44641.1	2	FALSE	FALSE	FALSE	3	1	5377.0	5	1	24718.0	1	0	2.15E+09
2	33233.7	11	TRUE	TRUE	TRUE	2	4	534.8	1	2	2703.7	22	1	5579.0
2	31395.1	6	TRUE	TRUE	TRUE	2	3	866.0	4	1	3742.0	22	1	5921.0
2	24954.8	5	FALSE	FALSE	FALSE	9	2	3789.3	4	1	11285.0	18	1	18659.0
3	64983.3	6	TRUE	TRUE	TRUE	3	3	2552.3	2	1	8528.0	5	1	21030.0
3	42709.7	3	FALSE	TRUE	TRUE	17	1	23062.0	3	1	31141.0	1	1	35435.0
4	56776.4	4	FALSE	TRUE	TRUE	22	1	732.0	20	1	9176.0	4	1	16755.0
4	42384.6	9	TRUE	TRUE	TRUE	4	4	1723.6	7	1	5649.0	16	2	9991.0
4	86780.8	4	FALSE	FALSE	FALSE	8	2	7450.7	2	1	18182.0	5	1	22628.0
4	111168.0	5	FALSE	FALSE	FALSE	5	1	12762.0	17	1	19317.0	1	1	24569.0
4	69501.1	3	FALSE	FALSE	FALSE	5	1	2996.0	9	1	31620.0	16	1	34056.0
5	69711.0	5	FALSE	TRUE	TRUE	20	1	20267.0	5	1	28606.0	7	1	32637.0
5	53399.8	3	FALSE	TRUE	TRUE	18	1	12121.0	14	1	20119.0	5	1	24789.0
5	87967.8	155	TRUE	TRUE	TRUE	5	153	0.95	12	1	6058.0	13	1	11310.0
5	80514.2	4	FALSE	FALSE	FALSE	24	1	9301.0	16	1	36290.0	18	1	38670.0
5	34705.0	5	FALSE	TRUE	TRUE	9	1	2971.0	7	1	4702.0	2	1	11752.0
6	809.0	0	FALSE	FALSE	FALSE	1	0	2.15E+09	2	0	2.15E+09	3	0	2.15E+09
6	2622.6	1	FALSE	FALSE	FALSE	13	1	17941.0	1	0	2.15E+09	2	0	2.15E+09
6	8628.2	7	FALSE	TRUE	TRUE	4	5	98.8	7	1	2685.0	6	1	3871.0
6	318.5	0	FALSE	FALSE	FALSE	1	0	2.15E+09	2	0	2.15E+09	3	0	2.15E+09
7	65883.6	7	FALSE	TRUE	TRUE	12	2	4639.0	7	1	8501.0	5	1	15594.0
7	21135.6	3	TRUE	TRUE	TRUE	7	1	5006.0	11	1	14076.0	14	1	36325.0
8	53389.9	4	FALSE	TRUE	TRUE	9	1	6124.0	13	1	13753.0	7	1	16986.0
8	82216.3	11	TRUE	TRUE	TRUE	8	4	598.8	23	2	2796.1	20	2	3617.1
8	49477.5	4	TRUE	TRUE	TRUE	8	3	725.4	22	1	15665.0	1	0	2.15E+09
9	64839.9	10	TRUE	TRUE	TRUE	9	2	1788.5	5	2	4682.2	4	2	5043.3
9	41790.1	7	FALSE	TRUE	TRUE	20	2	550.0	11	1	21887.0	5	1	24988.0
9	39692.1	2	TRUE	TRUE	TRUE	9	1	5802.0	5	1	40044.0	1	0	2.15E+09

Features2d RAM Classifiers

This experiment also builds on results obtained in Section 4.1.1 and the methodology discussed in Section 3.4.3. Unlike the *cv::ml* file-based classifiers discussed in the preceding section, the results discussed in this section were obtained from experiments that make use of classifiers loaded into random access memory (RAM). The FLANN based classifiers implemented in OpenCV`s *cv::FlannBasedMatcher* class can be used for both floating point and binary descriptors. The output of this experimental procedure is the calculation of the similarity index (SI) which is then used to classify leaf and stem samples.

Table 4.9. A snippet of 30 of the 103 test results obtained from running test images through a SIFT KDTreeIndexParams FLANN based in-memory classifier.

SIFT KDTreeIndexParams (1) - Load Time: 302.9s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	3075.61	2	FALSE	TRUE	TRUE	11	1	96.7	1	1	124.5	0	0	2.15E+09
1	3100.69	3	FALSE	FALSE	FALSE	16	1	141.0	23	1	142.8	10	1	146.9
2	1905.63	1	FALSE	FALSE	FALSE	7	1	136.1	0	0	2.15E+09	0	0	2.15E+09
2	1568.62	1	FALSE	FALSE	FALSE	9	1	58.3	0	0	2.15E+09	0	0	2.15E+09
3	3539.05	1	TRUE	TRUE	TRUE	3	1	149.3	0	0	2.15E+09	0	0	2.15E+09
3	2379.93	1	TRUE	TRUE	TRUE	3	1	163.8	0	0	2.15E+09	0	0	2.15E+09
4	2468.81	1	FALSE	FALSE	FALSE	7	1	75.2	0	0	2.15E+09	0	0	2.15E+09
5	4584.35	81	TRUE	TRUE	TRUE	5	81	0.018	0	0	2.15E+09	0	0	2.15E+09
6	652.239	4	FALSE	FALSE	FALSE	4	4	2.9	0	0	2.15E+09	0	0	2.15E+09
8	2899.27	1	TRUE	TRUE	TRUE	8	1	166.4	0	0	2.15E+09	0	0	2.15E+09
8	4078.64	1	TRUE	TRUE	TRUE	8	1	82.6	0	0	2.15E+09	0	0	2.15E+09
8	2522.03	1	TRUE	TRUE	TRUE	8	1	89.8	0	0	2.15E+09	0	0	2.15E+09
9	3433.92	2	TRUE	TRUE	TRUE	9	1	55.4	12	1	81.4	0	0	2.15E+09
10	1208.43	1	TRUE	TRUE	TRUE	10	1	117.7	0	0	2.15E+09	0	0	2.15E+09
10	1030.79	2	TRUE	TRUE	TRUE	10	2	20.3	0	0	2.15E+09	0	0	2.15E+09
10	396.713	5	TRUE	TRUE	TRUE	10	5	2.3	0	0	2.15E+09	0	0	2.15E+09
10	451.1	3	TRUE	TRUE	TRUE	10	3	9.3	0	0	2.15E+09	0	0	2.15E+09
10	388.7	4	TRUE	TRUE	TRUE	10	4	7.9	0	0	2.15E+09	0	0	2.15E+09
11	4794.2	3	FALSE	TRUE	TRUE	9	1	126.0	11	1	146.4	16	1	198.1
11	5007.9	1	FALSE	FALSE	FALSE	13	1	98.9	0	0	2.15E+09	0	0	2.15E+09
11	5161.2	2	FALSE	TRUE	TRUE	17	1	113.2	11	1	179.2	0	0	2.15E+09
12	1235.1	1	FALSE	FALSE	FALSE	20	1	156.8	0	0	2.15E+09	0	0	2.15E+09
12	1523.6	24	TRUE	TRUE	TRUE	12	24	0.20	0	0	2.15E+09	0	0	2.15E+09
13	3206.7	1	TRUE	TRUE	TRUE	13	1	135.5	0	0	2.15E+09	0	0	2.15E+09
13	4780.7	2	FALSE	FALSE	FALSE	18	1	162.0	8	1	166.4	0	0	2.15E+09
13	2638.8	2	FALSE	FALSE	FALSE	15	1	35.8	12	1	113.3	0	0	2.15E+09
14	2981.7	32	TRUE	TRUE	TRUE	14	30	0.14	8	1	71.8	20	1	159.1
14	2705.6	28	TRUE	TRUE	TRUE	14	27	0.16	11	1	87.3	0	0	2.15E+09
15	384.1	2	TRUE	TRUE	TRUE	15	2	8.3	0	0	2.15E+09	0	0	2.15E+09
16	6239.7	2	TRUE	TRUE	TRUE	16	1	79.8	22	1	156.0	0	0	2.15E+09

Table 4.9 shows results obtained from running test images through a SIFT KDTreeIndexParams, FLANN based in-memory classifier then extracting the similarity index in the custom matching algorithm. A value of 1 was used for the KDTreeIndexParams parameter. Appendix F shows results of other parameters considered and how index 1 gave the best results. The SIFT classifier initial load time was 302.9s, a value comparable to the file-based classifiers. For the unified dataset (leaves and stems), average calculation time per test sample was 2.5s. The FLANN based classifier showed considerable loss with 55.55% of the test images failing to

match to any sample in the unified dataset. Of the remaining 45.45%, the classifier correctly matched 60% of the test samples i.e. the classifier found the correct match at number one. 66.7% of the samples were correctly classified in the top 5 matches and the same amount were correctly classified in the top 10 matches. For the stem dataset, 64.9% of stem samples correctly matched at number one, 67.74% test samples were correctly classified in the top 5 and the same value was true for test samples classified in the top 10. For the leaf dataset, 37.8% of leaf samples correctly matched at number one, in the top 5 as well as in the top 10.

Table 4.10 shows test results obtained from running test images through a SIFT LinearIndexParams, FLANN based in-memory classifier then extracting the similarity index in the custom matching algorithm using the LinearIndexParams parameter. Only 25% of test images could be matched to one or more samples in the dataset therefore the SIFT algorithm using the LinearIndexParams parameter was extremely lossy and no further analysis was conducted on it.

Table 4.10. A snippet of 30 of the 103 test results obtained from running test images through a SIFT LinearIndexParams FLANN based in-memory classifier.

SIFT LinearIndexParams - Load Time: 258.2s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	56709.0	1	FALSE	FALSE	FALSE	11	1	96.7	0	0	2.15E+09	0	0	2.15E+09
1	70328.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	47830.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	34150.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	32268.1	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	26029.6	1	FALSE	FALSE	FALSE	9	1	58.3	0	0	2.15E+09	0	0	2.15E+09
3	69102.3	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	44961.1	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	67059.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	41447.7	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	86090.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	112046	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	71329.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	72089.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	54867.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	88793.2	79	TRUE	TRUE	TRUE	5	79	1.5	0	0	2.15E+09	0	0	2.15E+09
5	81342.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	35391.7	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	812.4	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	2622.7	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	8156.1	4	FALSE	FALSE	FALSE	4	4	11.6	0	0	2.15E+09	0	0	2.15E+09
6	318.9	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	67790.3	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	20712.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	55534.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	75422.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	44318.9	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	62202.2	2	TRUE	TRUE	TRUE	9	1	55.4	12	1	81.4	0	0	2.15E+09
9	39294.2	1	FALSE	FALSE	FALSE	5	1	158.0	0	0	2.15E+09	0	0	2.15E+09
9	38626.3	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09

Table 4.11 shows test results obtained from running test images through a SURF KDTreeIndexParams, FLANN based in memory classifier then extracting the similarity index in the custom matching algorithm. A value of 1 was used for the KDTreeIndexParams parameter. Appendix F shows various options explored. The SURF classifier initial load time was 304.6s. For the unified dataset (leaves and stems), average calculation time per test sample was 5.5s. 35.33% of the test images failed to match to any sample in the unified dataset. Of the remaining 64.66%, the classifier correctly matched 58% of the test samples i.e. the classifier found the correct match

at match number one. 61.5% of the samples were correctly classified in the top 5 matches and the same percentage in the top 10 matches. For the stem dataset, 57% of stem samples correctly matched at number one. 60.5% test samples were correctly classified in the top 5 and the same value was true for test samples classified in the top 10. For the leaf dataset, 58.33% of leaf samples correctly matched at number one, in the top 5 as well as in the top 10.

Table 4.11. A snippet of 30 of the 103 test results obtained from running test images through a SURF KDTreeIndexParams FLANN based in memory classifier.

SURF KDTreeIndexParams (1) - Load Time: 304.6s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	5841.9	1	FALSE	FALSE	FALSE	9	1	0.107	0	0	2.15E+09	0	0	2.15E+09
2	4355.8	4	TRUE	TRUE	TRUE	2	3	0.024	16	1	0.089	0	0	2.15E+09
4	14290.4	2	TRUE	TRUE	TRUE	4	2	0.022	0	0	2.15E+09	0	0	2.15E+09
4	5493.2	1	FALSE	FALSE	FALSE	16	1	0.087	0	0	2.15E+09	0	0	2.15E+09
5	5275.1	2	FALSE	FALSE	FALSE	7	1	0.080	9	1	0.174	0	0	2.15E+09
5	8643.2	73	TRUE	TRUE	TRUE	5	73	1.95E-05	0	0	2.15E+09	0	0	2.15E+09
5	7409.1	1	FALSE	FALSE	FALSE	17	1	0.189	0	0	2.15E+09	0	0	2.15E+09
5	4608.1	3	TRUE	TRUE	TRUE	5	2	0.017	11	1	0.086	0	0	2.15E+09
6	6467.4	5	TRUE	TRUE	TRUE	6	5	0.001	0	0	2.15E+09	0	0	2.15E+09
6	3607.3	1	TRUE	TRUE	TRUE	6	1	0.011	0	0	2.15E+09	0	0	2.15E+09
6	2278.8	1	FALSE	FALSE	FALSE	5	1	0.090	0	0	2.15E+09	0	0	2.15E+09
6	4595.5	2	TRUE	TRUE	TRUE	6	1	0.016	17	1	0.0188	0	0	2.15E+09
7	4290.7	1	FALSE	FALSE	FALSE	15	1	0.085	0	0	2.15E+09	0	0	2.15E+09
8	6093.3	3	TRUE	TRUE	TRUE	8	2	0.020	20	1	0.127	0	0	2.15E+09
9	7201.0	2	FALSE	TRUE	TRUE	22	1	0.078	9	1	0.115	0	0	2.15E+09
9	5642.6	2	TRUE	TRUE	TRUE	9	2	0.017	0	0	2.15E+09	0	0	2.15E+09
10	3173.7	2	TRUE	TRUE	TRUE	10	2	0.022	0	0	2.15E+09	0	0	2.15E+09
10	2415.8	4	TRUE	TRUE	TRUE	10	4	0.007	0	0	2.15E+09	0	0	2.15E+09
10	1908.4	15	TRUE	TRUE	TRUE	10	15	0.0002	0	0	2.15E+09	0	0	2.15E+09
10	2516.2	4	TRUE	TRUE	TRUE	10	3	0.005	6	1	0.047	0	0	2.15E+09
10	1816.5	6	TRUE	TRUE	TRUE	10	6	0.001	0	0	2.15E+09	0	0	2.15E+09
11	9373.5	1	FALSE	FALSE	FALSE	8	1	0.151	0	0	2.15E+09	0	0	2.15E+09
12	2740.7	2	FALSE	FALSE	FALSE	11	1	0.097	13	1	0.123468	0	0	2.15E+09
12	4182.2	44	TRUE	TRUE	TRUE	12	43	5.47E-05	5	1	0.097	0	0	2.15E+09
12	4255.1	2	TRUE	TRUE	TRUE	12	2	0.023	0	0	2.15E+09	0	0	2.15E+09
12	3141.7	1	FALSE	FALSE	FALSE	11	1	0.174	0	0	2.15E+09	0	0	2.15E+09
13	7097.6	2	FALSE	FALSE	FALSE	17	1	0.080	24	1	0.091	0	0	2.15E+09
13	9860.6	2	FALSE	FALSE	FALSE	8	1	0.055	17	1	0.103	0	0	2.15E+09
13	6622.7	1	FALSE	FALSE	FALSE	22	1	0.067	0	0	2.15E+09	0	0	2.15E+09
14	5164.5	15	TRUE	TRUE	TRUE	14	14	0.000	7	1	0.120	0	0	2.15E+09

Table 4.12. A snippet of 30 of the 103 test results obtained from running test images through a SURF LinearIndexParams FLANN based in-memory classifier.

SURF LinearIndexParams - Load Time: 266.4s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	68394.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	66364	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	54712.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	51936.1	1	FALSE	FALSE	FALSE	16	1	0.089	0	0	2.15E+09	0	0	2.15E+09
2	40078.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	37784.9	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	59817.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	33797.4	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	79373.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	69223.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	62004.3	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	168251	1	TRUE	TRUE	TRUE	4	1	0.042	0	0	2.15E+09	0	0	2.15E+09
4	60396	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	69815.7	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	60865	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	98040.1	70	TRUE	TRUE	TRUE	5	70	0.001	0	0	2.15E+09	0	0	2.15E+09
5	85963.9	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	51575.4	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	74713.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	40348.9	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	23777.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	49388.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	81959.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	48939.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	72381.9	2	TRUE	TRUE	TRUE	8	2	0.040	0	0	2.15E+09	0	0	2.15E+09
8	84426.4	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	59641.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	82812.6	3	FALSE	TRUE	TRUE	8	1	0.069	22	1	0.0786228	9	1	0.115571
9	59762.5	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	63343	1	TRUE	TRUE	TRUE	9	1	0.046	0	0	2.15E+09	0	0	2.15E+09

Table 4.12 shows test results obtained from running test images through the SURF LinearIndexParams, FLANN based in-memory classifier then extracting the similarity index in the custom matching algorithm using the LinearIndexParams parameter. Only 38% of test images could be matched to one or more samples in the dataset therefore the SURF algorithm using the LinearIndexParams parameter is extremely lossy. No further analysis was conducted for this algorithm.

Table 4.13. A snippet of 30 of the 103 test results obtained from running test images through the ORB5000 LshIndexParams FLANN based in memory classifier.

ORB5000 Binary LshIndexParams (12, 30, 0) - Load Time: 448.5s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	1288.5	26	FALSE	FALSE	FALSE	13	3	4.3	21	3	4.370	22	3	4.407
1	1164.9	30	FALSE	TRUE	TRUE	3	7	0.623	1	6	1.0	11	2	7.750
1	865.1	16	FALSE	FALSE	TRUE	4	2	6.5	13	2	8.625	3	2	9.625
2	540.9	11	FALSE	TRUE	TRUE	9	1	5.0	18	2	5.125	23	2	8.125
2	457.8	12	TRUE	TRUE	TRUE	2	3	3.7	11	1	6.0	18	2	8.750
2	421.4	7	FALSE	TRUE	TRUE	22	2	5.7	2	1	25.0	9	1	29.0
3	897.9	22	FALSE	TRUE	TRUE	9	1	1.0	3	4	1.593	10	3	4.740
3	867.1	35	TRUE	TRUE	TRUE	3	12	0.182	1	3	2.222	12	2	3.0
4	1073.0	50	TRUE	TRUE	TRUE	4	12	0.146	18	4	1.609	11	5	1.696
4	1048.5	26	TRUE	TRUE	TRUE	4	6	0.750	7	5	0.936	9	4	1.578
4	872.5	28	FALSE	FALSE	FALSE	11	3	3.851	9	1	4.0	18	2	6.250
4	1137.1	13	FALSE	TRUE	TRUE	2	2	6.125	17	2	8.625	4	1	19.0
4	856.2	16	FALSE	FALSE	TRUE	9	3	3.037	7	3	3.518	5	2	6.875
5	682.5	17	FALSE	FALSE	TRUE	23	2	6.625	7	1	7.0	10	1	9.0
5	668.3	17	FALSE	FALSE	FALSE	16	4	2.015	3	2	9.375	4	2	11.250
5	741.6	245	TRUE	TRUE	TRUE	5	231	0.0	3	1	5.0	18	2	11.375
5	734.2	18	FALSE	TRUE	TRUE	23	2	7.5	8	2	8.250	21	2	9.875
5	522.5	14	FALSE	FALSE	FALSE	24	1	3.0	10	1	12.0	9	2	12.250
6	486.1	2	TRUE	TRUE	TRUE	6	2	1.0	0	0	2.15E+09	0	0	2.15E+09
6	399.9	7	FALSE	TRUE	TRUE	4	1	3.0	6	1	4.0	22	3	4.185
6	296.1	2	TRUE	TRUE	TRUE	6	1	19.0	23	1	66.0	0	0	2.15E+09
6	309.0	2	TRUE	TRUE	TRUE	6	2	1.125	0	0	2.15E+09	0	0	2.15E+09
7	847.0	27	TRUE	TRUE	TRUE	7	6	1.041	9	1	2.0	10	2	3.0
7	662.6	8	FALSE	FALSE	FALSE	17	1	2.0	13	2	4.750	8	2	6.625
8	947.8	32	TRUE	TRUE	TRUE	8	7	0.218	9	5	0.792	21	1	2.0
8	824.3	23	TRUE	TRUE	TRUE	8	3	0.851	22	1	1.0	11	4	2.546
8	714.5	22	TRUE	TRUE	TRUE	8	5	0.536	13	5	1.712	6	1	2.0
9	1079.9	19	TRUE	TRUE	TRUE	9	3	0.1851	4	1	2.0	12	2	3.250
9	893.0	26	FALSE	TRUE	TRUE	24	2	0.25	8	5	0.680	9	4	1.953
9	916.5	35	TRUE	TRUE	TRUE	9	8	0.133	11	6	1.111	4	3	2.814

Table 4.13 shows test results obtained from running test images through the ORB5000 Binary LshIndexParams, FLANN based in memory classifier then extracting the similarity index in the custom matching algorithm. The classifier used LshIndexParams with values (12, 30, 0). Appendix F shows various options explored. The ORB5000 classifier initial load time was 448.5s. For the unified dataset (leaves and stems), average calculation time per test sample was 0.49s. The classifier showed a high degree of data preservation with 98.6% of the images successfully matched to one or

more samples in the dataset. The classifier correctly matched 51% of the test samples i.e. the classifier found the correct match at number one. 77% of the samples were correctly classified in the top 5 and 84% were correctly classified in the top 10 matches. For the stem dataset, 44.8% of stem samples correctly matched at number one, 72% of the test samples were correctly classified in the top 5 and 83.6% were correctly classified in the top 10. For the leaf dataset, 62.5% of leaf samples correctly matched at number one. 87.5% of the samples appeared in the top 5 and in the top 10 matches.

Table 4.14. A snippet of 30 of the 103 test results obtained from running test images through the AKAZE Binary LshIndexParams FLANN based in memory classifier.

AKAZE Binary LshIndexParams (12, 33, 0) - Load Time: 476.8s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	604.6	9	FALSE	TRUE	TRUE	22	2	21.5	1	1	34.0	10	1	48.0
1	577.0	10	FALSE	FALSE	FALSE	10	2	10.875	9	2	15.75	21	1	24.0
1	513.3	5	FALSE	FALSE	FALSE	10	1	24.0	24	1	51.0	9	1	65.0
2	523.2	5	FALSE	TRUE	TRUE	5	2	5.5	2	1	39.0	22	1	45.0
2	487.9	4	TRUE	TRUE	TRUE	2	3	3.666	21	1	26.0	0	0	2.15E+09
2	506.5	3	TRUE	TRUE	TRUE	2	1	32.0	3	1	95.0	6	1	111.0
3	580.0	3	TRUE	TRUE	TRUE	3	1	18.0	12	1	34.0	9	1	51.0
3	486.5	8	TRUE	TRUE	TRUE	3	4	3.375	4	1	8.0	9	1	54.0
4	543.9	4	TRUE	TRUE	TRUE	4	2	10.25	19	1	19.0	9	1	83.0
4	507.3	3	TRUE	TRUE	TRUE	4	1	56.0	7	1	76.0	9	1	81.0
4	552.6	3	FALSE	FALSE	FALSE	12	1	44.0	13	1	72.0	7	1	80.0
4	704.1	17	FALSE	FALSE	FALSE	17	3	6.814	9	2	19.875	11	2	20.5
4	551.0	6	TRUE	TRUE	TRUE	4	1	10.0	2	2	10.5	11	1	35.0
5	534.6	2	FALSE	FALSE	FALSE	4	1	17.0	10	1	98.0	0	0	2.15E+09
5	487.8	2	FALSE	FALSE	FALSE	11	1	43.0	13	1	83.0	0	0	2.15E+09
5	541.1	123	TRUE	TRUE	TRUE	5	121	0.002	23	1	76.0	13	1	93.0
5	502.3	2	FALSE	FALSE	FALSE	16	1	20.0	21	1	36.0	0	0	2.15E+09
5	471.9	3	FALSE	TRUE	TRUE	23	1	4.0	5	1	17.0	22	1	73.0
6	460.6	1	TRUE	TRUE	TRUE	6	1	33.0	0	0	2.15E+09	0	0	2.15E+09
6	445.0	2	TRUE	TRUE	TRUE	6	2	5.25	0	0	2.15E+09	0	0	2.15E+09
6	409.7	1	FALSE	FALSE	FALSE	23	1	35.0	0	0	2.15E+09	0	0	2.15E+09
6	435.3	2	FALSE	FALSE	FALSE	15	1	13.0	18	1	44.0	0	0	2.15E+09
7	499.8	6	TRUE	TRUE	TRUE	7	2	20.0	22	1	22.0	8	1	30.0
7	496.5	4	FALSE	TRUE	TRUE	11	1	24.0	7	1	40.0	2	1	41.0
8	485.9	4	FALSE	TRUE	TRUE	10	1	19.0	8	1	27.0	11	1	62.0
8	514.9	9	TRUE	TRUE	TRUE	8	2	9.25	7	2	13.25	5	1	61.0
8	476.5	3	FALSE	FALSE	FALSE	9	1	33.0	18	1	51.0	22	1	70.0
9	539.8	11	FALSE	FALSE	FALSE	17	4	6.359	7	2	16.625	4	1	34.0
9	495.5	4	TRUE	TRUE	TRUE	9	1	46.0	7	1	61.0	12	1	69.0
9	512.8	6	FALSE	TRUE	TRUE	8	1	4.0	23	2	10.125	24	2	16.125

Table 4.14 shows test results obtained from running test images through the FLANN based in memory classifier then extracting the similarity index in the custom matching algorithm. The classifier used LshIndexParams with values (12, 33, 0). The AKAZE classifier initial load time was 476.8s. For the unified dataset (leaves and stems), average calculation time per test sample was 0.5s. The AKAZE classifier also showed a high degree of data preservation with 99% of the test images matching to at least one sample. The classifier correctly matched 44% of the test samples i.e. the classifier found the correct match at match number one. 61% of the samples were correctly classified in the top 5 matches and the same percentage in the top 10 matches. For the stem dataset, 44% of stem samples correctly matched at number one. 62.7% test samples were correctly classified in the top 5 and the same value was true for test samples classified in the top 10. For the leaf dataset: 41.2% of leaf samples correctly matched at number one. 58.8% of the samples appeared in the top 5 as well as in the top 10.

4.1.3 Keypoint Analysis Summary

Image-to-Image Analysis

The image-to-image analysis exposed some inadequacies of comparisons conducted on a sample-to-sample basis. Working with geometrically similar but different objects means keypoint variations are more pronounced. Match rates in these cases are lower than when comparing an object with its affine transposed version. Using number of keypoints as a segregation technique is effective in eliminating false positives but it also has its drawbacks. A scaled image will produce different number of keypoints to its original image for example. If this keypoint variance is more than the set threshold this will cause the keypoint variance segregation test discussed in Section 4.1.1 (Subsection *Second Neighbour Ratio Test and Image-to-Image Custom Matching Algorithm Analysis*) to fail.

Scaling and Rotation Analysis

SIFT, ORB and AKAZE showed the greatest resiliency to rotation having a matching accuracy ≥65% from 0 to 2π radian angles of rotation. Daisy had the worst rotation invariance performance compared to other algorithms. Scaling analysis showed string descriptor-based algorithms had the greatest stability, with SIFT and SURF having a matching accuracy of approximately 65% and a variance of ±10% across scaling factors. ORB had a more linear response to scaling however it showed to be more susceptible to scaling than the string-based descriptors. AKAZE also showed good scaling resiliency with an accuracy of approximately 65% however the matching accuracy variation across scaling factors was much higher, averaging at approximately 45% accuracy.

Computation

Computational expense in the image-to-image analysis showed SIFT and SURF on average outperformed other algorithms for samples with low keypoint array sizes however their performance with OpenCV`s features2d and *cv::ml* classifiers showed otherwise, with SIFT and SURF giving the worst performance, approximately taking one minute to compute a match. ORB500 performed the best with high keypoint array size samples in the image-to-image analysis however this performance can be attributed to limitation of the N value used to return keypoints extracted. As expected, the larger the value of N used to return the number of quality matches the more

expensive the computation [14 p. 2, 16 p. 3]. A value of between 1000-5000 gives the best trade-off between computational speed and computed features returned. ORB5000 showed the best computational performance with the classifiers.

Classifier Analysis

As mentioned in 4.1.2, the *cv::ml* namespace classifier namespace can only be used with string-based descriptors hence only SIFT and SURF descriptors could be analysed. Compared to *cv::FlannBasedMatcher* classifiers, *cv::ml* file-based classifiers are much slower, taking an average of one minute to compute matches for a test image across the dataset. The *cv::ml* classifiers use a brute force search to match a sample to the trained model. *cv::FlannBasedMatcher* classifiers on the other hand use optimised search methods which on average execute faster than a brute force search [43]. The *cv::FlannBasedMatcher* class` worst performing classifier was the SURF KDTreeIndexParams1 implementation which took 5.5s and the best performance was from the ORB5000 - LshIndexParams(12, 30, 0) classifier which took 0.49s to compute a match. Although magnitudes slower, the *cv::ml* SIFT and SURF implementations performed better than the *cv::FlannBasedMatcher* algorithm implementations with regards to accuracy.

The results shown in Table 4.15 are grouped according to the datasets under investigation namely unified, stems and leaves. The two best (highlighted in green) and two worst (highlighted in red) performances of each column group per dataset are highlighted. The cv::ml file-based algorithms and the FlannBasedMatcher based algorithms utilising LinearIndexParams were the most computationally expensive with the worst (SURF) taking 66.6s to compute a match. The binary based descriptors using LshIndexParams gave the best computational performance with ORB5000-Lsh (12, 30, 0) computing a match in 0.49s. LinearIndexParams also showed to be lossy by failing to match a high number of test images and therefore were excluded from succeeding tests. Cv::ml SURF gave the best performance with the stem dataset followed by the ORB5000 algorithm. The two algorithms also performed better than other algorithms for the leaf dataset with ORB5000 giving the best performance in that category. SURF`s performance with the STEM dataset makes it suitable as the primary descriptor for STEM`s however it`s relatively high computational requirements (66.6 seconds to compute a match) make it not ideal for real time applications. ORB5000, although slightly less accurate than SURF for the STEM dataset, is a suitable candidate for STEM classification as well.

Table 4.15. A collation of results obtained from the keypoint analysis.

OpenCV namespace/class	Computation Time (s)	Unmatched percentage %	No. 1 Match %	Top 5 Match %	Top 10 Match %
Unified Dataset					
ml					
SIFT	43.7	0	52	74	74
SURF	66.6	0	60.4	82.35	89.22
FlannBasedMatcher					
SIFT- LinearIndexParams	44.6	75	-	-	-
SIFT- KDTreeIndexParams1	2.5	55.55	60	66.7	66.7
SURF- LinearIndexParams	64.8	62	-	-	-
SURF- KDTreeIndexParams1	5.5	35.33	58	61.5	61.5
ORB5000-Lsh(12, 30, 0)	0.49	1.4	51	77	84
AKAZE-Lsh(12, 33, 0)	0.5	1	44	61	61
Stems Dataset					
ml					
SIFT	-	-	56.7	77.6	77.6
SURF	-	-	70.3	87.5	92.2
FlannBasedMatcher					
SIFT- LinearIndexParams	-	-	-	-	-
SIFT- KDTreeIndexParams1	-	-	64.9	67.74	67.74
SURF- LinearIndexParams	-	-	-	-	-
SURF- KDTreeIndexParams1	-	-	57	60.5	60.5
ORB5000-Lsh(12, 30, 0)	-	-	44.8	72	83.6
AKAZE-Lsh(12, 33, 0)	-	-	44	62.7	62.7
Leaves Dataset					
ml					
SIFT	-	-	41.2	61.7	61.7
SURF	-	-	47.1	79.4	88.2
FlannBasedMatcher					
SIFT- LinearIndexParams	-	-	-	-	-
SIFT- KDTreeIndexParams1	-	-	37.8	37.8	37.8
SURF- LinearIndexParams			-	-	-
SURF- KDTreeIndexParams1	-	-	58.33	58.33	58.33
ORB5000-Lsh(12, 30, 0)	-	-	62.5	87.5	87.5
AKAZE-Lsh(12, 33, 0)	-	-	41.2	58.8	58.8

Keypoint Similarity Index Effectiveness

The range of SI values is dependent on the descriptor used. Using the ORB5000 results, the observations in Tables 4.16 and 4.17 were made.

Table 4.16. A snippet of the SI variance values obtained for correct matches.

Sample ID	Correct # 1 Match	Match 1	Match 1 SI	Match 2	Match 2 SI	Variance (%)
15	TRUE	15	1.42188	17	7.0	392.30
24	TRUE	24	1.01563	17	6.375	527.68
15	TRUE	15	1.85185	23	14.0	656.0
15	TRUE	15	0.87200	17	7.0	702.75
17	TRUE	17	0.12426	23	1.0	704.76
9	TRUE	9	0.13281	11	1.111	736.59
23	TRUE	23	0.05515	8	0.475	761.69
9	TRUE	9	0.18518	4	2.0	980.0
4	TRUE	4	0.14641	18	1.609	999.21
3	TRUE	3	0.18287	1	2.222	1115.19
10	TRUE	10	0.50926	22	7.375	1348.18
24	TRUE	24	0.15393	18	4.333	2715.03
16	TRUE	16	0.02879	13	1.0	3373.41
10	TRUE	10	0.17205	1	27.0	15593.02
10	TRUE	10	0.14500	22	23.0	15761.63
14	TRUE	14	0.00647	9	2.851	43956.19
18	TRUE	18	0.00171	8	2.421	141711.2
12	TRUE	12	0.00273	23	6.125	224309.1
14	TRUE	14	0.00187	17	4.851	259171.4
5	TRUE	5	0.00038	3	5.0	1299603.9
15	TRUE	15	1.42188	17	7.0	392.30
24	TRUE	24	1.01563	17	6.375	527.68
15	TRUE	15	1.85185	23	14.0	656.0
15	TRUE	15	0.87200	17	7.0	702.75
17	TRUE	17	0.12426	23	1.0	704.76
9	TRUE	9	0.13281	11	1.111	736.59

Table 4.17. A snippet of the SI variance values obtained for incorrect matches.

Sample ID	Correct # 1 Match	Match 1	Match 1 SI	Match 2	Match 2 SI	Variance (%)
16	FALSE	17	4.556	16	4.592	0.81
21	FALSE	11	2.953	12	3.0	1.58
13	FALSE	24	4.296	13	4.370	1.72
11	FALSE	13	1.328	8	2.703	103.48
21	FALSE	8	1.0	21	2.078	107.81
4	FALSE	9	3.037	7	3.518	15.85
2	FALSE	9	5.0	18	5.125	2.5
20	FALSE	11	9.750	2	10.0	2.56
1	FALSE	13	4.259	21	4.370	2.60
11	FALSE	17	2.0	11	2.064	3.2
4	FALSE	11	3.851	9	4.0	3.84
1	FALSE	4	6.5	13	8.625	32.69
19	FALSE	9	1.0	19	1.333	33.33
6	FALSE	4	3.0	6	4.0	33.33
21	FALSE	7	3.453	5	4.629	34.07
14	FALSE	11	27.0	24	37.0	37.03
16	FALSE	4	10.125	11	10.625	4.93
18	FALSE	16	1.937	17	2.718	40.32
4	FALSE	2	6.125	17	8.625	40.81
23	FALSE	22	1.576	24	2.250	42.76
5	FALSE	23	6.625	7	7.0	5.66
16	FALSE	21	3.046	22	4.851	59.23
3	FALSE	9	1.0	3	1.593	59.35
1	FALSE	3	0.623	1	1.0	60.28
20	FALSE	9	1.848	20	3.156	70.79
16	FALSE	17	4.555	16	4.592	0.812

The results obtained in Tables 4.16 and 4.17 showed that a SI variance greater than 300% between match 1 and match 2 resulted in a positive match 92% of the time. In some of the remaining cases, a qualitative analysis of the matched result showed that the match marked at number 1 was structurally like the test sample.

4.2 Moments Analysis

Moments offer a more structural approach to classification of objects, as the derivation of moments takes into consideration the shape of a blob as opposed to derivation of descriptors per pixel for instance [35, 38].

Moment analysis will focus on the following:

- Effect of image scaling and rotation on matching accuracy. These are the affine transformations anticipated to most likely be the commonly encountered by the FLORA system.
- Percentage match accuracy the Hu moments classifier with and without the custom matching algorithm.
- Hu moments extraction, descriptor computation and descriptor comparison execution speed.
- Analysis on the effectiveness of the moments custom matching algorithm and the resultant similarity index.

The data splitting technique used for the moments classifier uses the same technique discussed in Section 3.4.1 A 4-fold strategy was employed which yielded the results in Table 4.18:

Table 4.18. Iterative training and testing over the four folds showed split 1 gave the best accuracy for moments analysis using Hu moment descriptors.

					Avg Accuracy
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	34%
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	22%
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	29%
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	32%
		Training Data		Testing Data	

Scaling and rotation analysis were conducted on the leaf and stem databases using a random sample of 5 leaf and 6 stem images from the respective databases.

4.2.1 Rotation Analysis

Each test image is rotated and then compared to its original version by calculating the L2-Norm (Euclidean distance) between the two samples.

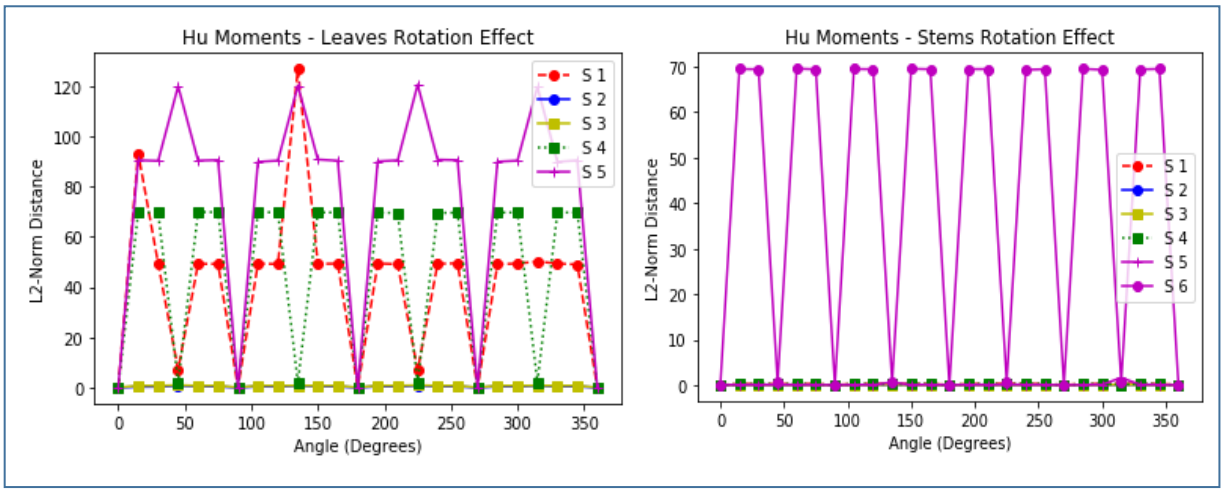


Figure 4.8. Moments rotation analysis of stems and leaves.

Figure 4.8 shows rotation of stems gave a more stable response, except for sample S6, while leaf rotation showed erratic and unreliable behaviour away from π/4 radian multiples across samples. Both Stems and leaves however showed accurate matching for multiples of π/4 radians.

4.2.2 Scaling Analysis

Each test image is scaled and then compared to its original version by calculating the L2-Norm (Euclidean distance) between the two samples.

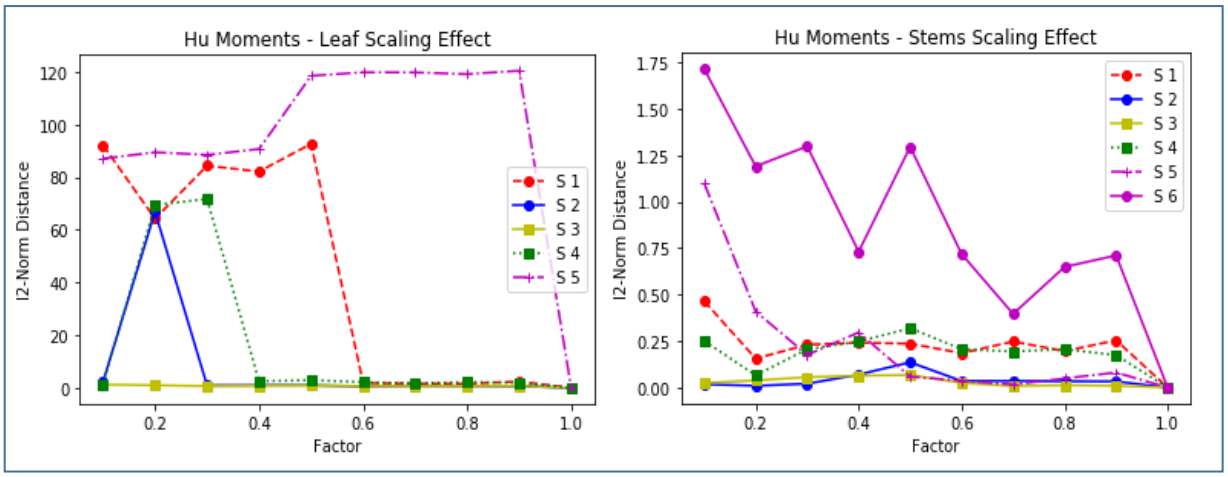


Figure 4.9. Moments analysis of scaling stems and leaves.

Figure 4.9 shows scaling of both stems and leaves gave stable responses for most of the test samples. Scaling of stems showed a constant response for all factors except for the sample S6. Leaf scaling also showed consistent matching with scaled images down to a factor of 0.6 for most of the test samples except sample S5. Beyond that the Euclidean distance response was erratic.

4.2.3 Match Accuracy and Comparison Speed

Experimentation on the matching accuracy focused on the effectiveness of using Hu moments for classification of leaves and stems as well as the qualitative and quantitative effect of using the custom matching algorithm. An analysis on the computational speed of extracting moment descriptors and using them for matching was also be conducted.

Table 4.19 shows results obtained from Hu moment comparison using a cv::ml-based k-NN classifier. The classifier load time is relatively fast taking 59ms to load. The unified dataset calculation time per sample was 26.7ms and the classifier gave a correct sample prediction accuracy of 30.3%. The Hu moments classifier correctly predicted a sample 35.3% of the time for the leaf dataset while correctly predicting a sample 40% of the time for the stem dataset.

Table 4.19. A snippet of 30 of the 103 results obtained from Hu moment comparison using a cv::ml-based k-NN classifier.

Hu Moments Raw Results – Classifier Load Time: 0.059s			
Sample ID	Calc Time (ms)	Prediction	Match
1	11	32.4939	FALSE
1	24	28.8836	FALSE
1	11	22.9765	FALSE
2	2	24.3029	TRUE
2	2	27.3658	TRUE
2	24	23.3429	FALSE
3	7	26.0042	FALSE
3	10	25.8684	FALSE
4	23	27.6191	FALSE
4	17	28.0786	FALSE
4	5	46.3041	FALSE
4	4	33.0052	TRUE
4	24	26.8239	FALSE
5	5	35.0307	TRUE
5	5	34.3992	TRUE
5	19	35.7966	FALSE
5	1	32.0345	FALSE
5	6	24.8511	FALSE
6	17	27.7438	FALSE
6	6	20.8380	TRUE
6	22	26.9641	FALSE
6	1	30.9745	FALSE
7	17	28.8176	FALSE
7	14	26.6962	FALSE
8	11	64.3705	FALSE
8	17	34.3823	FALSE
8	8	28.7563	TRUE
9	17	33.7106	FALSE
9	8	27.7951	FALSE
9	9	24.3413	TRUE

Table 4.20 shows test results obtained from running test images through the cv::ml based classifier then extracting the similarity index in the custom matching algorithm. The classifier initial load time was 0.061s. For the unified dataset, average calculation time per test sample was 28ms and the classifier correctly matched 35% of the test samples i.e. the classifier found the correct match at match number one. 73% of the samples were correctly classified in the top 5 matches and 85% of the samples correctly classified in the top 10 matches. For the stem dataset, 27% of stem samples correctly matched at number one, 61.2% test samples were correctly classified in the

top 5 and 77.6% of the samples were correctly classified in the top 10 matches. For the leaf dataset, 47% of leaf samples correctly matched at number one, 91.2% of the samples appeared in the top 5 and 94% of the same samples correctly classified in the top 10 matches.

Table 4.20. A snippet of 30 of the 103 test results obtained from running test images through a Hu moments cv::ml based classifier.

Hu Moments Classifier with Matching Algorithm - Load Time: 0.061s													
		Correct Match											
Sample ID	Calc Time (ms)	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	33.0	FALSE	FALSE	TRUE	2	5	0.164	5	3	0.552	7	2	0.599
1	28.7	FALSE	FALSE	TRUE	5	4	0.086	24	2	0.142	11	2	0.236
1	28.6	FALSE	FALSE	FALSE	5	5	0.009	11	4	0.009	13	3	0.017
2	31.2	FALSE	TRUE	TRUE	5	4	0.056	2	4	0.080	11	2	0.189
2	32.6	TRUE	TRUE	TRUE	2	3	0.312	16	4	0.386	5	1	0.460
2	29.4	FALSE	TRUE	TRUE	18	5	0.158	16	4	0.210	17	3	0.309
3	31.4	FALSE	TRUE	TRUE	14	4	0.021	19	4	0.021	7	2	0.039
3	30.2	FALSE	TRUE	TRUE	10	12	0.004	3	3	0.092	7	2	0.205
4	27.8	FALSE	FALSE	FALSE	23	1	0.171	12	2	0.176	1	2	0.219
4	30.5	FALSE	TRUE	TRUE	24	5	0.043	20	4	0.067	4	3	0.111
4	31.7	FALSE	FALSE	FALSE	2	5	0.042	5	4	0.051	24	3	0.130
4	39.0	FALSE	TRUE	TRUE	20	4	0.034	23	3	0.043	8	3	0.087
4	28.0	FALSE	FALSE	FALSE	2	5	0.066	5	4	0.095	24	3	0.149
5	35.0	TRUE	TRUE	TRUE	5	3	0.032	11	4	0.039	7	3	0.054
5	30.0	FALSE	TRUE	TRUE	19	10	0.003	5	3	0.026	9	2	0.103
5	37.5	FALSE	TRUE	TRUE	17	3	0.026	5	3	0.029	11	3	0.030
5	32.4	FALSE	FALSE	TRUE	17	5	0.006	23	4	0.025	1	1	0.029
5	28.7	FALSE	TRUE	TRUE	13	7	0.003	11	5	0.007	6	3	0.015
6	30.8	FALSE	TRUE	TRUE	17	9	0.005	6	5	0.015	23	4	0.021
6	25.4	TRUE	TRUE	TRUE	6	9	0.001	11	3	0.022	5	3	0.026
6	35.0	FALSE	TRUE	TRUE	19	9	0.013	22	1	0.055	7	3	0.144
6	35.6	TRUE	TRUE	TRUE	6	4	0.012	12	4	0.018	17	3	0.025
7	32.6	FALSE	FALSE	TRUE	17	4	0.007	12	3	0.009	9	2	0.020
7	29.0	FALSE	FALSE	FALSE	18	7	1.621	4	3	3.201	22	3	3.624
8	29.2	FALSE	FALSE	FALSE	22	4	0.013	11	3	0.023	13	3	0.027
8	33.4	FALSE	FALSE	FALSE	17	5	0.010	18	5	0.015	20	3	0.029
8	29.1	TRUE	TRUE	TRUE	8	10	0.004	7	3	0.025	5	3	0.056
9	37.8	FALSE	TRUE	TRUE	11	4	0.007	17	3	0.015	9	3	0.020
9	28.4	FALSE	FALSE	TRUE	13	6	0.004	8	2	0.012	7	3	0.022
9	25.8	FALSE	TRUE	TRUE	11	4	0.044	22	3	0.067	9	2	0.079

4.2.4 Moments Analysis Summary

Scaling and Rotation Analysis

Stem response to rotation and scaling was much better than that of leaves. Stem images on average have a higher pixel density than leaf images because stem samples are much larger than individual leaves. This would explain the more stable stem response to scaling. Erratic behaviour is observed with stem sample S6 (shown below) for both scaling and rotation. Sample S6 is the only stem sample tested with Length x Width dimensions less than 1000pixels ×1000pixels. Leaf analysis on rotation and scaling also shows that the worst performance was from the leaf sample S5, which also had the lowest Length × Width dimensionality compared to other samples.

Figure 4.10. Left: Leaf sample S5. Right: Stem sample S6. Leaf sample S5, with dimensions 723px × 1993px. Stem sample S6 (pre-flowering Leucospermum reflexum var. luteum) with dimensions 792px × 1721px.

Analysis shows that for reliable scaling and rotation response it is preferable to keep sample dimensionality above 1000px × 1000px.

Match Accuracy and Comparison Speed Analysis

The Hu moments classifier is much faster than the keypoints matching classifiers, loading in under 100ms and computing a match, on average, in 28ms. It`s first match accuracy is however lower with the classifier correctly matching only 30.3% of the test samples at match number one. The accuracy of the classifier with the custom matching algorithm is not any higher for first matches with a match rate of 35%. The use of the Hu moments custom matching algorithm however significantly increases the chances of finding a match by extracting the top 5 and top 10 matches. As can be seen from

Table 4.20, 73% of the test samples in the unified dataset where correctly classified in the top 5 matches, while the leaves dataset gave a 91.2% match rate in the same regard. The custom matching algorithm, although not improving the first match accuracy significantly, increased the probability of finding a match within a bound list of matches. A qualitative analysis of the results also shows that the custom matching algorithm improved the quality of results returned even if the match was false. Returned results show a lot more structural similarity as opposed to when the Hu moments classifier is used with no matching algorithm.

Figure 4.11. Test results without the use of the custom matching algorithm. Left: A Leucospermum Glabrum pincushion stem sample. Right: Leucodendron teretifolium (Greyton

In Figure 4.11, with testing results without the use of the custom matching algorithm, the Leucospermum Glabrum stem sample matches to Leucodendron teretifolium in the unified dataset at match number 1. With the algorithm (Figure 4.12), the classifier successfully matches to the correct genus even though it is unable to pinpoint the specific specie. The two samples visibly have structural similarities.

Figure 4.12. Results with the custom matching algorithm Left: A Leucospermum Glabrum pincushion stem sample. Right : Luecospermum Grandiflorum (Grey leaf fountain pincushion).

Moments Similarity Index Effectiveness

The Hu moments SI values showed to be less discriminatory than the keypoint matching SI when comparing the first and second returned matches. The Hu moments classifier uses descriptors derived from the outline of an object in an image. Results obtained in Section 4.2.3 and the discussion conducted in Section 5.1 show that the Hu moments classifier is much better at clustering similar matches than the keypoint classifier. Subsequently, the Hu moments classifier will invariably have a higher probability of clustering structurally similar matches together even if the matches are not the correct ones. This would explain the low SI variances between match 1 and match 2 for some of the correct matches shown in Table 4.21. Table 4.22 shows that incorrect matches had a variance less than 60% however as shown in Table 4.21, 14 of the 26 (54%) correct matches also had a variance less than 60%.

Table 4.21. A snippet of the SI variance values for Hu moments correct matches.

Sample ID	Correct # 1 Match	Match 1	Match 1 SI	Match 2	Match 2 SI	Variance (%)
12	TRUE	12	0.02141	6	0.02198	2.70
12	TRUE	12	0.00770	9	0.00796	3.34
18	TRUE	18	0.04086	16	0.04526	10.77
17	TRUE	17	0.01017	16	0.01131	11.13
13	TRUE	13	0.01452	11	0.01712	17.85
5	TRUE	5	0.03223	11	0.03907	21.23
2	TRUE	2	0.31241	16	0.38662	23.76
10	TRUE	10	0.00918	19	0.01187	29.28
13	TRUE	13	0.01348	22	0.01822	35.16
10	TRUE	10	0.01055	3	0.01513	43.35
23	TRUE	23	0.01086	17	0.01591	46.44
23	TRUE	23	0.02118	17	0.03164	49.37
6	TRUE	6	0.01217	12	0.01871	53.66
21	TRUE	21	0.01264	7	0.01973	55.99
15	TRUE	15	0.00102	18	0.00179	74.26
16	TRUE	16	0.00820	18	0.01824	122.32
22	TRUE	22	0.01045	21	0.03372	222.42
22	TRUE	22	0.00788	13	0.02617	232.03
14	TRUE	14	0.00565	13	0.01932	242.10
19	TRUE	19	0.00253	6	0.00924	264.57
15	TRUE	15	0.00123	18	0.00459	272.79
16	TRUE	16	0.02724	18	0.10278	277.25
19	TRUE	19	0.00240	5	0.00998	315.88
17	TRUE	17	0.00289	6	0.01331	360.34
24	TRUE	24	0.00736	17	0.03672	398.86
8	TRUE	8	0.00452	7	0.02507	454.32

Table 4.22. A snippet of the SI variance values for Hu moments incorrect matches.

Sample ID	Correct # 1 Match	Match 1	Match 1 SI	Match 2	Match 2 SI	Variance (%)
17	FALSE	23	0.15351	20	0.15409	0.38
3	FALSE	14	0.02107	19	0.02116	0.40
1	FALSE	5	0.00903	11	0.00933	3.33
4	FALSE	23	0.17111	12	0.17687	3.36
18	FALSE	17	0.03841	16	0.03998	4.07
23	FALSE	17	0.01502	16	0.01607	6.97
24	FALSE	6	0.01563	24	0.01703	8.92
14	FALSE	6	0.01091	11	0.01217	11.55
5	FALSE	17	0.02640	5	0.02962	12.20
21	FALSE	20	0.03384	2	0.03852	13.82
17	FALSE	23	0.01643	17	0.01916	16.61
12	FALSE	23	0.01757	17	0.02104	19.74
16	FALSE	18	0.03331	17	0.04002	20.15
4	FALSE	2	0.04262	5	0.05175	21.42
18	FALSE	23	0.02093	18	0.02622	25.28
4	FALSE	20	0.03491	23	0.04380	25.44
23	FALSE	16	0.23298	17	0.30359	30.31
20	FALSE	18	0.02257	17	0.02995	32.71
13	FALSE	7	0.01123	19	0.01491	32.75
2	FALSE	18	0.15864	16	0.21096	32.98
7	FALSE	17	0.00740	12	0.00987	33.49
18	FALSE	16	0.07592	17	0.10639	40.12
17	FALSE	16	0.50547	23	0.71688	41.82
2	FALSE	5	0.05600	2	0.08078	44.23
8	FALSE	17	0.01058	18	0.01529	44.47
4	FALSE	2	0.06606	5	0.09588	45.14

Although the custom matching algorithm used on Hu moments does not significantly improve the accuracy, it serves one further purpose of allowing ranking of returned results (as shown in Table 4.20), a task not possible with the raw Hu moments algorithm.

4.3 Hashing Analysis

Experimental procedure follows the steps outlined in Section 3.4.5. Analysis carried out by extracting hash values and comparing across samples showed that the hash algorithm`s inter species values did not show great variation overall (shown in Table 4.23).

Table 4.23. A sample of the results obtained from raw extraction of hash values from training samples in different species.

Hash Analysis				
Sample ID	**Average Hash**	**PHash**	**Colour Moment Hash**	**Block Mean Hash**
1	16	31	27.89	60
1	13	32	35.35	47
2	23	31	41.7	68
2	7	28	51.85	41
3	3	29	101.15	47
3	8	33	433.7	49
4	16	29	20.45	47
4	16	29	22.18	60
5	10	39	157.8	54
5	13	30	167.0	58
6	2	38	81.3	52
6	2	37	75.7	56

The results in Table 4.23 showed little variation across species with the hash algorithms. Descriptor variability is essential for correct classification. Colour Moment hash gave the best variability across species. pHash showed the least variability across samples and therefore is unsuitable for stem and leaf classification purposes. Given the results obtained in Table 4.23, analysis then focused on the Colour Moment and Average hashing algorithms.

The matching of hash values is conducted using a FLANN based k-NN classifier. Average hash expects CV_8U (unsigned char) data and computes Hamming distance while Colour Moment hash expects CV_64F (double) data and matches by computing L2-Norm distance. Average hash therefore makes use of a FLANN based matcher that uses LshIndexParams and the Colour Moment hash makes use of a KDTreeIndexParams based FLANN matcher.

The splitting technique used for the hash algorithms classifier is the same as that discussed in Section 3.4.1. A 4-fold strategy was employed which yielded results in Table 4.24:

Table 4.24. Iterative training and testing using the Average hash algorithm with the 4-fold strategy.

					Avg Accuracy
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	4%
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	2%
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	7%
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	3%
		Training Data		Testing Data	

Iterative training and testing over the 4 folds showed split 1 gave the best accuracy using the Average hash algorithm. Average matching accuracy with the hash algorithm was less than 10% for all splits. k-NN classifiers were then trained and running test samples through the classifiers gave the results in Table 4.25.

Table 4.25. A snippet of 30 of the 103 results obtained from running test samples through an Average Hash FLANN based Hamming distance classifier.

Average Hash Classifier - Load Time: 74.1s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	21.1632	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	1
1	21.1722	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	1
1	21.0307	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	1
2	23.5728	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2
2	20.3131	1	FALSE	FALSE	FALSE	17	1	1	0	0	2.15E+09	0	0	2
2	19.987	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2
3	22.3947	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	3
3	19.0096	1	FALSE	FALSE	FALSE	10	1	0	0	0	2.15E+09	0	0	3
4	20.7419	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	4
4	20.9928	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	4
4	19.7558	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	4
4	24.9091	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	4
4	23.3412	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	4
5	25.6214	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	5
5	23.6003	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	5
5	29.5554	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	5
5	29.2208	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	5
5	22.0256	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	5
6	24.8223	1	FALSE	FALSE	FALSE	14	1	4	0	0	2.15E+09	0	0	6
6	19.0384	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	6
6	21.4284	1	TRUE	TRUE	TRUE	6	1	0	0	0	2.15E+09	0	0	6
6	28.5388	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	6
7	30.3447	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	7
7	21.8229	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	7
8	26.2864	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	8
8	26.7205	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	8
8	24.3701	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	8
9	29.5416	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	9
9	23.2448	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	9
9	21.3952	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	9

Results presented in Table 4.25 were obtained from running test samples through an Average hash FLANN based Hamming distance classifier. The results show unreliability with the Average Hash algorithm with the classifier giving a 9% match rate. Table 4.26 shows results obtained from running test samples through a Colour Moment hash FLANN based Hamming distance classifier. Results show unreliability with the Colour Moment hash algorithm with the classifier giving a 17% match rate.

Table 4.26. A snippet of 30 of the 103 results obtained from running test samples through a Colour Moment Hash FLANN based Hamming distance classifier.

Colour Moment Hash Classifier - Load Time: 74.1s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	55.5149	1	FALSE	FALSE	FALSE	13	1	0.000438	0	0	2.15E+09	0	0	2.15E+09
1	58.1794	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	46.2024	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	54.7686	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	49.0531	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	46.6547	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	54.5863	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	47.6313	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	48.9432	1	TRUE	TRUE	TRUE	4	1	0.000186	0	0	2.15E+09	0	0	2.15E+09
4	55.8235	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	48.2608	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	50.5121	1	TRUE	TRUE	TRUE	4	1	0.000367	0	0	2.15E+09	0	0	2.15E+09
4	54.4963	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	63.1914	1	TRUE	TRUE	TRUE	5	1	0.000180	0	0	2.15E+09	0	0	2.15E+09
5	51.8229	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	55.144	1	TRUE	TRUE	TRUE	5	1	0.000256	0	0	2.15E+09	0	0	2.15E+09
5	57.3899	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	52.1296	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	52.2326	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	45.9412	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	49.6591	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	58.1044	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	62.3774	1	TRUE	TRUE	TRUE	7	1	0.000401	0	0	2.15E+09	0	0	2.15E+09
7	51.052	1	FALSE	FALSE	FALSE	5	1	0.000529	0	0	2.15E+09	0	0	2.15E+09
8	52.5369	1	TRUE	TRUE	TRUE	8	1	0.000259	0	0	2.15E+09	0	0	2.15E+09
8	57.8048	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	51.3457	1	FALSE	FALSE	FALSE	7	1	0.000771	0	0	2.15E+09	0	0	2.15E+09
9	58.9221	1	FALSE	FALSE	FALSE	13	1	9.62E-05	0	0	2.15E+09	0	0	2.15E+09
9	52.7043	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	48.9929	1	TRUE	TRUE	TRUE	9	1	0.000232	0	0	2.15E+09	0	0	2.15E+09

The initial tests conducted on the Colour Moment and Average hash algorithms when used on the stems and leaves dataset showed the algorithms to be unreliable and therefore unusable for accurate matching. No further analysis on the effect of scaling or rotation was then conducted having ascertained the unsuitability of these algorithms in matching stems and leaves.

5 Hybrid Prototype Design and Analysis

Based on results obtained from the experimental procedures outlined in Chapter 3, a hybrid classification system was designed which incorporates the various algorithms and techniques investigated. The overall system diagram is shown below.

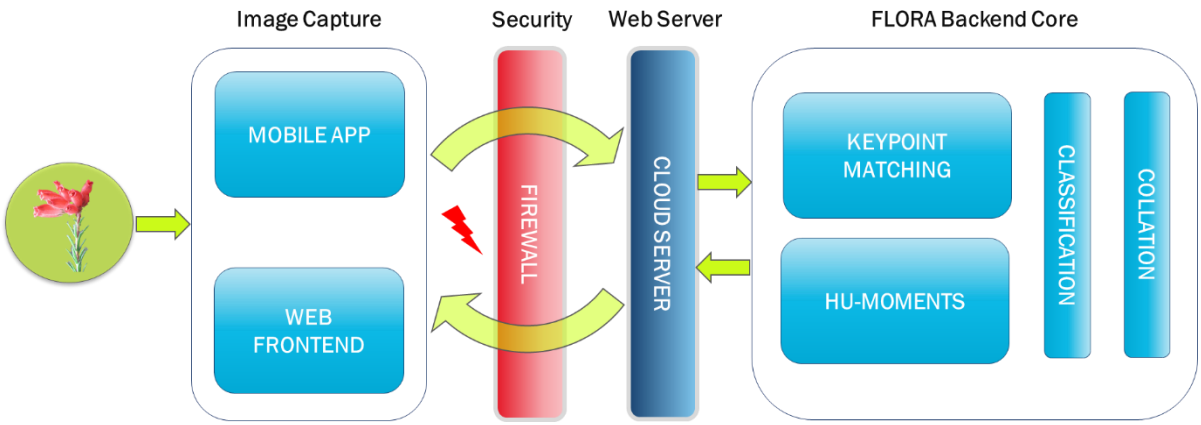


Figure 5.1. Overall system diagram for the FLORA fynbos leaf and stem identification platform.

Figure 5.1 shows how the FLORA backend core interacts with various components to achieve matching in real world applications. Multiple image input avenues to the FLORA classification core are available. Identification is carried out by an ORB5000 Keypoint matching – Hu moments classifier using a custom matching algorithm that extracts SI values used to rank potential matches. The input image source could be a public facing website, a mobile based application or direct image upload into the backend core. As part of this thesis, a .NET MVC front end website was built to facilitate access to the FLORA core backend as shown in Figure 5.2. The FLORA web portal comprises of a simple drag and drop interface where users can add images to query. Returned results are shown in the right-hand panel displayed from the most likely match at the top to the least likely at the bottom.

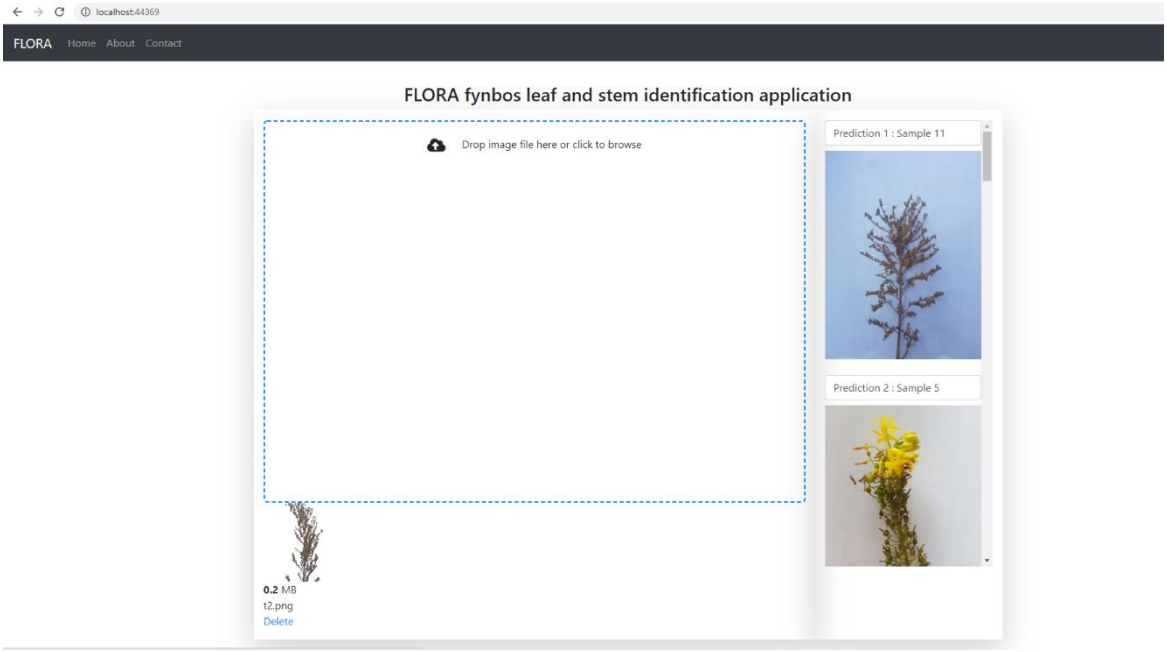


Figure 5.2. The FLORA web portal interface.

The FLORA backend core makes use of a keypoint matching-Hu moments-based hybrid classifier. The operational process flow of the FLORA backend is illustrated in Figure 5.3.

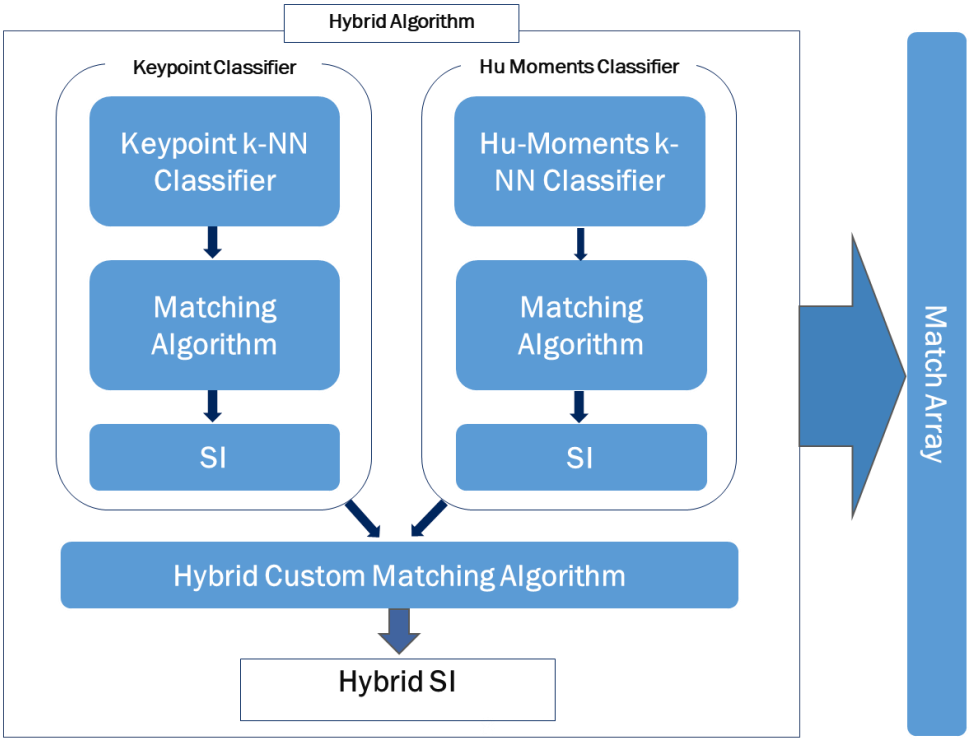


Figure 5.3. The FLORA backend hybrid algorithm brings together keypoint matching and Hu moments to provide an SI value used in ranking potential matches.

The pseudo-code, Listing 5.1, shows how the hybrid algorithms custom matching function calculates the hybrid SI values used to rank matches.

```
Declare CombinedArray;
FOR each keypointElement in KeypointsArray
       DECLARE sampleId to keypointElement.SampleId;
       SET CombinedArray[sampleId + 1].SI to
         (IF keypointElement.SI is not valid
              0
          ELSE
              Size of CombinedArray / (index of keypointElement in
              KeypointsArray + 1)
          END IF
         )
         FOR each huMomentsElement in MomentsArray
              IF huMomentsElement.SampleId == sampleId
                 IF first match
                    ADD 70% of huMomentsElement.SI to
                        CombinedArray[sampleId + 1].SI
                 ELSE
                    ADD huMomentsElement.SI to CombinedArray[sampleId + 1].SI
                 END IF
              END IF
          END FOR
  END FOR
```

Listing 5.1. Hybrid algorithms custom matching function pseudo-code.

The algorithm takes into consideration Hu moments` low first match accuracy which was 16% lower than the ORB5000 keypoint classifiers first match accuracy as shown

in Tables 4.13 and 4.20. Experimentation showed that weighting the Hu moments first match contributions based on the accuracy ratio with keypoint matching gives optimal results. The weighting is therefore calculated as shown in Equation 5.1.

$$\left(\text{Hu moments first match accuracy} \Big/ \text{ORB Keypoint first match accuracy}\right) \qquad \textit{Eq 5.1}$$

From Tables 4.13 and 4.20, the first match accuracy for Hu moments is 35% and that for ORB5000 is 51%. Using Equation 5.1 with these values yields a weighting value of 0.686. This is then rounded to the nearest tenth yielding a weighting value of 0.70 (70% of the Hu moments first match value). Subsequent match accuracy for the Hu moments classifier is comparable to the ORB5000 keypoint classifier hence both algorithms contribute the same weight to the hybrid SI. The Hu moments and keypoint SI values depicted in the above pseudo-code are on different scales. Normalisation is achieved by ordering the elements of each array based on the SI value and using the index to calculate the hybrid SI.

5.1 Custom Matching Algorithm Design

The custom matching algorithms used throughout the thesis were designed using results and observations discussed in Chapter 4. The Hu moments and ORB5000 custom matching algorithms provide the SI values that directly result in the hybrid custom matching algorithm and SI. This section analyses the relationships established between various factors and how these resulted in the constituent Hu moments and ORB5000 SI values and resultantly in the hybrid SI. Experimental observation indicated to a relationship existing between match accuracy and number of occurrences of a sample which were present in the returned results array.

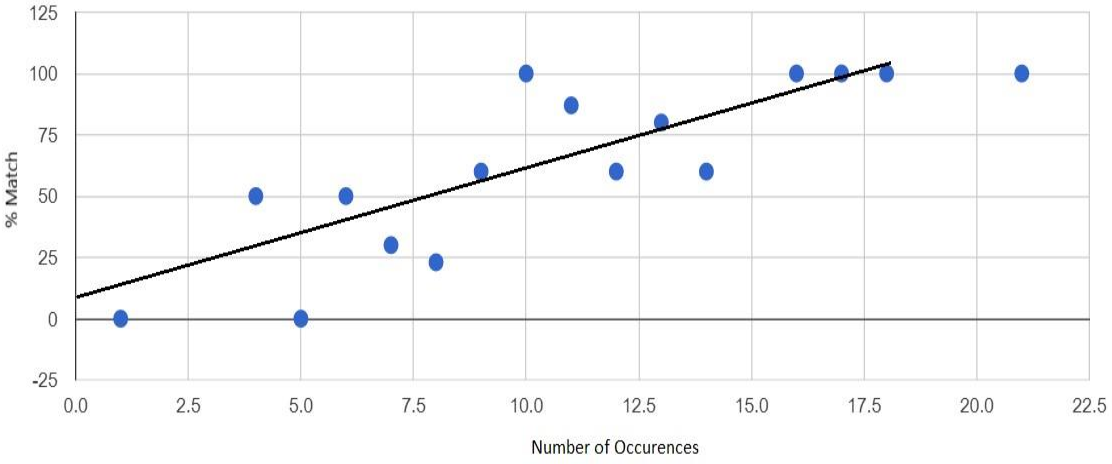


Figure 5.4. Graph of percentage match versus number of occurrences.

Figure 5.4 is plotted by grouping returned ORB5000 results by number of occurrences and finding the percentage of correct matches. Drawing a line of best fit shows a linear relationship can be inferred to exist between number of occurrences and the percentage match.

The ORB5000 and Hu moments SI values are defined to be a number between 0 and ∞ with an SI of zero indicating a complete match and ∞ indicating no match. Based on this definition and using the results obtained above, an inverse relationship is observed to exist between the SI value and number of occurrences of a sample (Also shown in Equation 3.3).

The keypoint results obtained in Section 4.1.2 and the Hu moment results obtained in Section 4.2.3 were also used to show that the distance returned from the k-NN classifier could also act as a discriminatory metric for the SI value. In general, the greater the distance of the match from the test sample, the less likely it is a match. The distance is also dependant on the type of algorithm used in the k-NN classifier. Plotting the average distance versus percentage match however showed that in all cases, a linear relationship exists between the average distance and the SI. Given distance of a match from a test sample is D_n {n=1,2…N} the average distance is defined as:

$$\frac{\sum_1^N D_n}{Total\ Number\ of\ Occurences} \qquad \textit{Eq 5.2}$$

Plotting the percentage match against the average distance from the test sample descriptors yields the below graph.

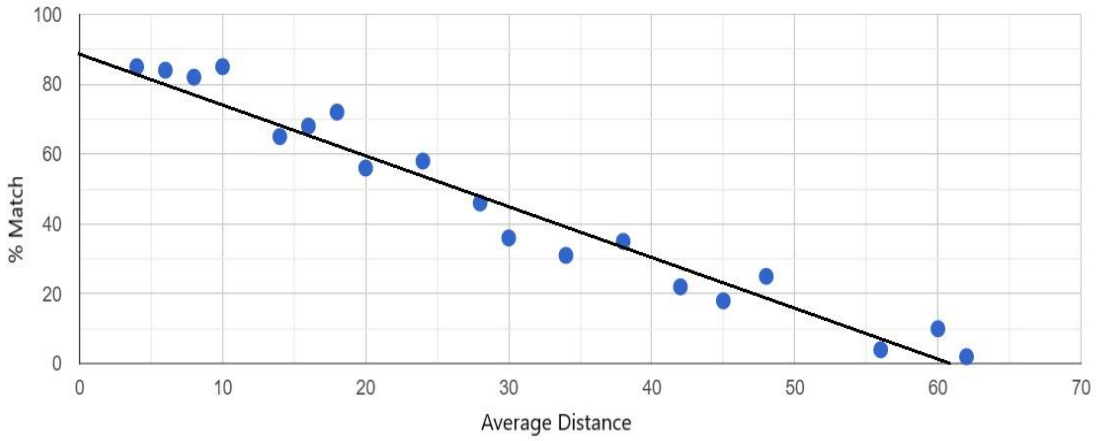


Figure 5.5. Graph shows the linear relationship that exists between percentage match and average distance.

Figure 5.14 shows the linear relationship that exists between percentage match and average distance obtained from the ORB5000 match results in 4.1.2. The graph shows the greater the distance from the test sample descriptors, the less likely the match is correct. From this it can be deduced that the SI value is directly proportional to the average distance (Also shown in Equation 3.3).

The hybrid SI, inversely to the Hu moments and ORB5000 SI, is defined to be a number between 0 and two times the size of the sample dataset, corresponding to the two SI values used to calculate the hybrid SI which individually contribute a maximum value equal to the size of the sample dataset. In the case of this dissertation this number is 48. Unlike the Hu moments and ORB5000 SI values, the greater the hybrid SI the more likely a sample is a match. The hybrid SI is extracted by considering the weighted index of a sample in the constituent Hu moments – keypoint result arrays as shown in the pseudo code in Listing 5.1.

5.2 Hybrid Algorithm Analysis

The experimental procedures carried out in Chapter 4 showed that keypoint matching classification gave the best matching accuracy out of the three chosen classification techniques explored in this thesis. Hu moments-based classifiers gave the best computational performance even though they lack in first match precision. This section will present an analysis of the results obtained from experimental procedures conducted on the hybrid algorithm focusing on the following:

- Match Accuracy.
- Computational speed.
- Effectiveness of the hybrid algorithm similarity index (SI).

5.2.1 Matching Accuracy and Computational Speed

This section focuses on the accuracy of the hybrid algorithm compared to the ORB5000 keypoint algorithm and the Hu moments algorithm. The hybrid classifier makes use of the custom matching algorithm discussed in Section 5.1.

Table 5.1. A snippet of 30 of the 103 test results obtained from running test images through the hybrid classifier.

Hybrid Classifier with Matching Algorithm - Load Time: 494.3s													
		Correct Match											
Sample ID	Calc Time (ms)	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	1755.4	FALSE	FALSE	FALSE	13	8	25.0	21	6	13.0	22	5	9.0
1	1583.3	FALSE	TRUE	TRUE	3	11	26.0	1	8	16.0	11	4	16.0
1	1010.1	FALSE	FALSE	TRUE	4	7	25.0	13	6	20.0	11	2	15.0
2	655.1	FALSE	TRUE	TRUE	9	5	27.0	2	4	18.0	18	6	13.0
2	546.1	TRUE	TRUE	TRUE	2	6	40.8	11	5	16.0	17	4	12.0
2	562.3	FALSE	TRUE	TRUE	22	7	25.0	2	5	18.0	17	3	14.0
3	1090.0	FALSE	TRUE	TRUE	9	5	27.0	3	8	18.0	5	6	10.0
3	1124.6	TRUE	TRUE	TRUE	3	24	36.0	1	6	15.0	7	4	14.0
4	1299.7	TRUE	TRUE	TRUE	4	13	26.0	23	3	17.8	11	7	14.0
4	1115.9	TRUE	TRUE	TRUE	4	11	32.0	7	9	14.0	9	7	9.0
4	1000.3	FALSE	FALSE	FALSE	11	8	30.0	2	3	19.8	5	4	16.0
4	1243.9	FALSE	TRUE	TRUE	2	6	26.0	17	5	15.0	4	4	14.0
4	1090.3	FALSE	FALSE	TRUE	9	8	27.0	5	5	20.0	7	7	13.0
5	907.2	FALSE	TRUE	TRUE	23	5	250.0	7	5	20.0	5	2	18.8
5	800.3	FALSE	FALSE	FALSE	16	14	25.0	3	5	14.0	4	4	10.0
5	932.1	TRUE	TRUE	TRUE	5	234	36.0	17	4	20.8	11	4	14.0
5	931.5	FALSE	TRUE	TRUE	23	7	36.0	17	2	19.8	8	6	13.0
5	663.7	FALSE	FALSE	FALSE	24	8	25.0	11	5	18.0	10	6	13.0
6	600.3	TRUE	TRUE	TRUE	6	11	36.0	1	5	4.0	2	4	3.0
6	494.9	TRUE	TRUE	TRUE	6	4	28.8	4	10	26.0	22	6	9.0
6	405.5	TRUE	TRUE	TRUE	6	10	28.0	23	2	13.0	1	3	3.0
6	447.4	TRUE	TRUE	TRUE	6	6	40.8	5	2	4.0	1	4	3.0
7	1035.4	TRUE	TRUE	TRUE	7	10	27.0	9	4	20.0	17	3	19.8
7	1174.9	FALSE	FALSE	FALSE	17	8	28.0	18	2	22.8	13	5	13.0
8	1340.5	TRUE	TRUE	TRUE	8	11	25.0	9	8	13.0	13	3	11.0
8	989.3	TRUE	TRUE	TRUE	8	8	25.0	17	5	22.8	18	3	16.0
8	858.7	TRUE	TRUE	TRUE	8	15	40.8	13	8	13.0	6	4	10.0
9	1360.9	TRUE	TRUE	TRUE	9	7	32.0	17	3	15.0	4	4	13.0
9	1042.8	FALSE	TRUE	TRUE	24	8	25.0	8	7	24.0	7	4	14.0
9	1023.5	TRUE	TRUE	TRUE	9	12	32.0	11	9	28.8	22	3	14.0

Table 5.1 shows the results obtained from tests run using the hybrid algorithm. Initial load time for the hybrid classifier was 494.3s. The unified dataset`s average calculation time per test sample was 0.78s. The classifier correctly matched 54% of the test samples i.e. the classifier found the correct match at match number one. 81% of the samples were correctly classified in the top 5 matches and 85% of the samples correctly classified in the top 10. Tests on the stem's dataset showed 46.3% of stem samples correctly matched at number one. 76.11% of stem test samples were correctly classified in the top 5 and 83% were correctly classified in the top 10.

The leaves dataset gave a 65% number one match accuracy and in 86% of cases, the correct match was in the in the top 5 as well as in the top 10.

The hybrid classifier performed only slightly better than the ORB5000 classifier with regards to matching accuracy as shown in Table 5.2. Although no significant improvement to matching accuracy was observed, qualitative analysis done in Section 5.2.2 shows that the use of the hybrid algorithm improves the clustering of returned matches. Table 5.2 below compares the matching accuracies of the different algorithms investigated in this thesis.

Table 5.2. A comparison of the hybrid algorithm, ORB5000 classifier and the Hu moments classifier.

	Computation Time (s)	Unmatched percentage	No. 1 Match %	Top 5 Match %	Top 10 Match %
Unified Dataset					
ORB5000	0.49	1.4	51	77	84
Hu Moments	0.028	0	35	73	85
Hybrid Classifier	0.78	0	54	81	85
Stems Dataset					
ORB5000	-	-	44.8	72	83.6
Hu Moments	-	-	27	61.2	77.6
Hybrid Classifier	-	-	46.3	76.11	83
Leaves Dataset					
ORB5000	-	-	62.5	87.5	87.5
Hu Moments	-	-	47	91.2	94
Hybrid Classifier	-	-	65	86	86

Table 5.2 shows that there is slight improvement with the unified dataset when the hybrid algorithm is used. The best performing algorithms for each dataset are highlighted in green. The first match accuracy for the leaves dataset also improved although subsequent match accuracy was slightly lower. Computation time for the hybrid algorithm is also higher than for the Hu moments and keypoint classifiers although within acceptable bounds for real time computation as the hybrid classifier returns results in under one second (requirement F6 in Section 1.5).

5.2.2 Hybrid SI Effectiveness

The hybrid SI is a floating-point number whose value lies between zero and two times the size of the sample dataset as indicated in Section 5.1. Each of the two constituent algorithms that make up the hybrid algorithm can contribute a maximum integer value of 24 to the hybrid SI. Using results from table 5.27, analysis showed that an SI value ≥30 yielded a positive first match rate of 72% and a top 5 match rate of 82%. Conversely, matches with an SI value <30 only yielded a first match rate of 18%.

Qualitative analysis of results returned with and without the custom matching algorithm also showed an increase in match clustering of structurally similar samples when the custom matching algorithm was used.

Table 5.3. Qualitative analysis test on Image 2 of Unnamed Sample 3.

Test Image	Match 1	Match 2	Match 3
	Without Custom Matching Algorithm		
	Sample 3	Sample 12	Sample 4
			
	With Custom Matching Algorithm		
	Sample 3	Sample 1	Sample 7
			

In Table 5.3, the results at the top were obtained from running the ORB5000 classifier. Although the correct match was identified, using the custom algorithm that incorporates the Hu moments returned results that are visibly structurally similar than without the classifier.

Table 5.4. Qualitative analysis test on Image 4 of Unnamed Sample 10.

Test Image	Match 1	Match 2	Match 3
	Without Custom Matching Algorithm		
	Sample 10	None	None
			
	With Custom Matching Algorithm		
	Sample 10	Sample 1	Sample 7
			

In Table 5.4, The ORB5000 classifier gave the correct first match result although could not identify any other similar matches. The custom matching algorithm, on top of identifying the correct match, was also able to provide other matches that share structural similarity with the correct match. Similar results were also obtained for image 1 of unnamed sample 12 in Table 5.5. The ORB5000 classifier was unable to correctly match the test sample. The custom matching classifier however correctly found the match at number 2. Structural similarities can be observed with the match identified as match 1.

Table 5.5. Qualitative analysis test on image 1 of unnamed sample 12.

Test Image	Match 1	Match 2	Match 3
	Without Custom Matching Algorithm		
	Sample 4	Sample 16	Sample 21
			
	With Custom Matching Algorithm		
	Sample 4	Sample 12	Sample 16
			

In Table 5.6, The test image is a stem sample of a Thompsons Phoenix, a member of the Leucospermum genus. Analysis without the custom matching algorithm was unable to correctly match the test image to the right sample. The returned result also showed structural deviance from the test sample. The custom matching algorithm was not only able to return the correct match at number 1, but it also matched the structurally similar Mimmetes Cucullatus at number 2, which belongs to the same Proteaceae family.

Table 5.6. Qualitative analysis test on a Thompsons Phoenix sample.

Test Image	Match 1	Match 2	Match 3
	Without Custom Matching Algorithm		
	Sample 6	Sample 17	Sample 22
			
	With Custom Matching Algorithm		
	Sample 18	Sample 16	Sample 6
			

5.2.3 Hybrid Results Analysis

Experimental results presented in Chapter 4 show that an ORB5000 based keypoint matching image classifier alone has a 77% matching capability of classifying test samples in the top 5 of returned results. The ORB5000 classifier could be used alone for matching however as the database grows, false positives become more likely with the introduction of more species from the same botanical taxonomical groups. It is therefore important to have a secondary mode of classification using a different matching technique that can be used as a results validity test. Unlike keypoint descriptors, Hu-moment descriptors are computed based on the overall geometric shape of a target object. This method of classification allows the hybrid algorithm to rely on the strengths of both keypoint matching and Hu moments resulting in more reliable matches as shown in Section 5.2.2.

Results presented in Section 5.2.1 show that this setup achieved a first match rate of 54% and in 81% of cases the correct species was in the top 5 of viable species returned. The system is currently not intended to work as a substitute for scientific

botanical classification and in most cases, it is not capable of discerning between species of the same family or, in the best case, genera. The system is designed to provide a list of viable matches ranked according to their SI values. With a comprehensive database of species from the same family, the FLORA system is just as likely to pick any of the species given they are structurally similar to each other as both Hu moments and the ORB based keypoint matching depend on descriptors derived from visual characteristics of a plants leaf or stem.

6 Conclusions and Future Work

6.1 Conclusion

This thesis presents the design and development of an object identification and classification algorithm capable of classifying fynbos stems and leaves. The output of this research was the design of a keypoint matching – Hu moments hybrid algorithm which gave a top 5 match accuracy of 81% as shown in Section 5.2.1. The concept of using a similarity index (SI) score for the purposes of match ranking was also introduced. The SI score allowed the hybrid algorithm to successfully return a ranked list of potential matches for a given sample input.

The objectives of this thesis were split into primary and secondary, with primary objectives being the core points of investigation. The first primary objective was to present literature on available object similarity and classification techniques. The literature review section presented identification and classification methods which were considered for this thesis. The other primary objectives were concerned with the experimental investigation of classification techniques suitable for leaf and stem classification as well as the conceptualisation and design of an affine invariant classification algorithm capable of botanical family or, best case, genus level recognition. Experimental procedures were outlined in Chapter 3 with corresponding experimental results presented in Chapter 4, and the designed hybrid algorithm presented in Chapter 5.

Experimentation done with keypoint matching showed that SIFT and SURF had the best responses to scaling and rotation however their relatively high computational cost with classifiers made them unsuitable based on the real-time functionality requirement F6, set for this thesis in Section 1.5. ORB5000 gave the best balance between speed of computation and accuracy. Hu moments, with only 7 bin descriptors, outperformed keypoint matching with regards to computation time, on average giving a match in 26.7ms. Accuracy over the leaf dataset was also higher than that obtained with the keypoint matching algorithms, achieving a 91.2% top 5 match rate. An observation that was expected given Hu-moment descriptors are primarily dependant on the outline of an object and leaves have a more defined structure than stems. Image hashing techniques discussed in Section 2.3.2 with experimental results presented in Section 4.3 showed a high degree of inaccuracy with the chosen algorithms giving a match

accuracy less than 20%, which was considered not useful for the purposes of this project.

As part of the secondary objectives, a web-based platform was designed to facilitate user access to the backend cores` functionality.

6.2 Future Work

Based on the results obtained over the course of this thesis and conclusions derived in Section 6.1, the following recommendations are made for future work and improvements to the FLORA project.

- Expand the fynbos image dataset. A larger dataset means more training material for the classifiers. This will facilitate for the comprehensive testing of the algorithms investigated in this thesis with a dataset that better mimics the real-world use case of the FLORA platform.
- Investigate the feasibility of re-implementing the algorithms to use a convolutional neural network (CNN). A CNN implementation could potentially be better suited for high volume data, which is what the FLORA platform will eventually consist of as the sample dataset expands.
- Design the FLORA platform to work with video. Multiple frames from a video could be used to improve accuracy of the algorithms used instead of the algorithms relying on single perspective images.

7 References

[1] S. Katz, “Fynbos Leaf-based Online Recognition Application,” University of Cape Town, 2011.

[2] J. Manning and P. Goldblatt, *Plants of The Greater Cape Floristic Region 1: The Core Cape Flora*, vol. 29. Pretoria: Strelitzia, 2012.

[3] N. Allsopp, J. Colville, and A. Verboom, *Fynbos: Ecology, Evolution, and Conservation of a Megadiverse Region*. Oxford: Oxford Scholarship Online, 2014.

[4] M. Rougeta, D. M. Richardson, R. MCowling, J. W. Loyd, and A. T. Lombard, “Current patterns of habitat transformation and future threats to biodiversity in terrestrial ecosystems of the Cape Floristic Region, South Africa,” *Biological Conservation - Elsevier*, vol. Volume 112, no. Issues 1–2, p. Pages 63-85, 2003, [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0006320702003956.

[5] A. M. Streich, K. A. Todd, and E. L. Specialist, “Classification and Naming of Plants,” Nebraska, 2014. [Online]. Available: https://extensionpublications.unl.edu/assets/pdf/ec1272.pdf.

[6] V. H. HEYWOOD, “Plant Classification,” *Nature*, vol. 221, no. 5177, pp. 293–293, 1969, doi: 10.1038/221293a0.

[7] M. C. Rutherford, L. Mucina, A. G. Rebelo, C. Boucher, and N. Helme, “Fynbos Biome,” *Vegetation Map of South Africa, Lesotho and Swaziland* , vol. 1, no. 1, pp. 1–219, 2006.

[8] N. C. Parkman, “A STRUCTURAL-FUNCTIONAL CLASSIFICA1ION OF THE FYNBOS VEGETATION OF THE BAIN’ S KLOOF AREA,” Cape Town, 1978. [Online]. Available: https://open.uct.ac.za/handle/11427/24966.

[9] E. Mchenry, “Lesson 2 : Plant Classification,” Wisconsin. [Online]. Available: http://www.ellenjmchenry.com/homeschool-freedownloads/lifesciences-games/documents/BotanySecondChapter_000.pdf.

[10] R. Rosselló-Móra, “Taxonomy,” *Taxonomy. In: Gargaud M. et al. (eds) Encyclopedia of Astrobiology. Springer, Berlin, Heidelberg*, 2011, [Online]. Available: https://link.springer.com/referenceworkentry/10.1007%2F978-3-642-11274-4_1562#howtocite.

[11] M. A. Ramone, “Evaluation of Machine Learning and Image Processing Algorithms Suitable for the Fynbos Identification – Building Up the Fynbos Online Recognition Application (FLORA) Literature Review on Fynbos,” University of Cape Town, 2014.

[12] M. A. Ramone, “FLORA – Fynbos Leaf Online Recognition Application,” University of Cape Town, 2013.

[13] K. Naidoo and S. Winberg, “GPU Acceleration of the Fynbos Leaf- based Online Recognition Application,” University of Cape Town, 2015.

[14] E. Karami, S. Prasad, and M. Shehata, “Image Matching Using SIFT , SURF , BRIEF and ORB : Performance Comparison for Distorted Images,” Canada, 2015. [Online]. Available: https://www.researchgate.net/publication/318194587_Image_Matching_Using_SIFT_SURF_BRIEF_and_ORB_Performance_Comparison_for_Distorted_Images.

[15] E. Tola, V. Lepetit, P. Fua, and S. Member, “DAISY : An Efficient Dense Descriptor Applied to Wide-Baseline Stereo,” *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 32, no. 5, pp. 1–17, 2010, doi: 10.1109/TPAMI.2009.77.

[16] R. Verma, “A Comparative Study of Various Types of Image Noise and Efficient Noise Removal Techniques,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 10, pp. 617–622, 2013, [Online]. Available: https://www.researchgate.net/publication/307545428_A_comparative_study_of_various_types_of_image_noise_and_efficient_noise_removal_techniques.

[17] R. M. Cowling, R. L. Pressey, M. Rouget, and A. T. Lombard, “A conservation plan for a global biodiversity hotspot - The Cape Floristic Region, South Africa,” *Biological Conservation*, vol. 112, no. 1–2, pp. 191–216, 2003, doi: 10.1016/S0006-3207(02)00425-1.

[18] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 1, no. 1, pp. 1–28, 2004, doi: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[19] E. Rublee, V. Rabaud, and K. Konolige, “ORB : an efficient alternative to SIFT or SURF,” *IEEE International Conference on Computer Vision*, vol. 1, no. 1, pp. 1–5, 2011, doi: 10.1002/art.38045.

[20] D. MARR and E. HILDRETH, “Theory of edge detection,” Boston, Massachusetts, 207, 1980.

[21] U. Sinha, “Convolutions,” *AI Shack*, 2016. https://aishack.in/tutorials/image-convolution-examples/ (accessed Aug. 04, 2020).

[22] H. Bay, A. Ess, T. Tuytelaars, and L. Vangool, “Speeded-Up Robust Features (SURF) (Cited by: 2272),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, doi: 10.1016/j.cviu.2007.09.014.

[23] A. Maetouq *et al.*, “Comparison of Hash Function Algorithms Against Attacks : A Review,” *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 8, pp. 98–103, 2018, [Online]. Available: https://thesai.org/Downloads/Volume9No8/Paper_13-Comparison_of_Hash_Function_Algorithms.pdf.

[24] A. Menezes, V. Oorschot, and S. Vanstone, *Applied Cryptography*, vol. 1. CRC Press, 1997.

[25] V. Monga and B. L. Evans, “Perceptual image hashing via feature points: Performance evaluation and tradeoffs,” *IEEE Transactions on Image*

Processing, vol. 15, no. 11, pp. 3452–3465, 2006, doi: 10.1109/TIP.2006.881948.

[26] A. Hadmi, W. Puech, B. Ait, E. Said, and A. A. Ouahman, “Perceptual Image Hashing,” Morocco, 2005. [Online]. Available: www.intechopen.com.

[27] N. Krawetz, “The hacker factor blog,” *The Hacker Factor Blog*, 2011. http://hackerfactor.com/blog/index.php%3F/archives/432-Looks-Like-It.html (accessed Oct. 03, 2019).

[28] ContentBlockchain, “TESTING DIFFERENT IMAGE HASH FUNCTIONS,” *Content Blockchain*, 2016. https://content-blockchain.org/research/testing-different-image-hash-functions/.

[29] C. Zauner, “Implementation and benchmarking of perceptual image hash functions,” University of Applied Sciences, 2010.

[30] B. Yang, F. Gu, and X. Niu, “Block mean value based image perceptual hashing,” Shenzhen, 2006. doi: 10.1109/IIH-MSP.2006.265125.

[31] F. Lef`ebvre, B. Macq, and J. Legat, “RASH : RAdon Soft Hash algorithm,” Belgium, 2002. [Online]. Available: https://www.eurasip.org/Proceedings/Eusipco/2002/articles/paper745.pdf.

[32] J. Radon, “On the Determination of Functions From Their Integral Values Along Certain Manifolds,” *IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. MI-5, NO. 4*, vol. 5, no. 4, pp. 170–176, 1986.

[33] B. Vukelić and M. Bača, “Comparison of RADIAL variance based and Maar-Hildreth operator perceptual image hash functions on biometric templates,” *Central European Conference on Information and Intelligent System*, vol. 1, no. 1, pp. 286–292, 2014.

[34] Z. Tang, Y. Dai, and X. Zhang, “Perceptual Hashing for Color Images Using Invariant Moments,” *Applied Mathematics & Information Sciences*, no. 2, 2012.

[35] M. K. Hu, “Visual Pattern Recognition by Moment Invariants,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962, doi: 10.1109/TIT.1962.1057692.

[36] S. Bettuzzi, “Chapter 1: Introduction to Image Moments,” *Advances in cancer research*, vol. 104, no. 1, pp. 1–8, 2009, doi: 10.1016/S0065-230X(09)04001-9.

[37] L. Kotoulas and I. Andreadis, “Image analysis using moments,” *Information Systems Journal*, vol. 1, no. 1, pp. 360–364, 2005, doi: 10.1109/TPAMI.2007.70709.

[38] D. Li, “Analysis of moment invariants on image scaling and rotation,” *Innovations in Computing Sciences and Software Engineering*, vol. 1, no. 1, pp. 415–419, 2010, doi: 10.1007/978-90-481-9112-3-70.

[39] S. Privett, “The Fynbos Hub,” *The Fynbos Hub*, 2011.

http://www.fynboshub.co.za/ (accessed Nov. 05, 2019).

[40] Z. Reitermanová, Annual Conference of Doctoral Students (19 2010.06.01-04 Prague), WDS'10 (19 2010.06.01-04 Prague), and Week of Doctoral Students 2010 (19 2010.06.01-04 Prague), "19th Annual Conference of Doctoral Students, WDS'10 'Week of Doctoral Students 2010,'" Prague, 2010. [Online]. Available: https://www.mff.cuni.cz/veda/konference/wds/proc/pdf10/WDS10_105_i1_Reitermanova.pdf.

[41] Y. Xu and R. Goodacre, "On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning," *Journal of Analysis and Testing*, vol. 2, no. 3, pp. 249–262, 2018, doi: 10.1007/s41664-018-0068-2.

[42] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," Vancouver, 2009. Accessed: Mar. 22, 2020. [Online]. Available: https://docs.opencv.org/2.4/modules/flann/doc/flann_fast_approximate_nearest_neighbor_search.html.

[43] OpenCV, "cv::FlannBasedMatcher Class Reference," *cv::FlannBasedMatcher Class Reference*, 2018. https://docs.opencv.org/4.3.0/dc/de2/classcv_1_1FlannBasedMatcher.html (accessed Oct. 14, 2019).

[44] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," Princeton, 2007.

[45] O. Anava and K. Y. Levy, "k*-Nearest Neighbors : From Global to Local," New York, 2017. [Online]. Available: https://papers.nips.cc/paper/2016/file/2c6ae45a3e88aee548c0714fad7f8269-Paper.pdf.

[46] S. B. Imandoust and M. Bolandraftar, "Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events : Theoretical Background," *International Journal of Engineering Research and Applications*, vol. 3, no. 5, pp. 605–610, 2013, [Online]. Available: https://www.ijera.com/papers/Vol3_issue5/DI35605610.pdf.

[47] OpenCV, "OpenCV Data Types," *OpenCV Documentation*, 2018. https://docs.opencv.org/3.4/d1/d1b/group__core__hal__interface.html#ga4a3def5d72b74bed31f5f8ab7676099c.

[48] ThamBlog, "Speed up image hashing of opencv(img_hash) and introduce color moment hash," 2016. https://qtandopencv.blogspot.com/search?q=hash (accessed Jun. 01, 2020).

[49] E. Klinger and D. Starkweather, "pHash : The open source perceptual hash library," *pHash.org*, 2011. http://www.phash.org/ (accessed Sep. 02, 2019).

[50] OpenCV, "OpenCV Image Hashing Algorithms," *OpenCV Documentation*,

2019. https://docs.opencv.org/4.3.0/d4/d93/group__img__hash.html (accessed Apr. 22, 2020).

Appendix A FLORA-E Hybrid Algorithm C++ Code

A.1 Hybrid Initialise Function

```
/***************Hybrid algorithm initialise function*********************/
static cv::FlannBasedMatcher matcher;
static cv::Ptr<KNearest> knn;
bool InitialiseHybridClassifiers(const char* imagePath, const char* filePath) {
	std::string image_path(imagePath);
	std::string train_path(filePath);

	if (knn == nullptr) {
		try {
			knn = Algorithm::load<KNearest>(train_path);
			train_keypoint_classifier_inmemory(image_path,  "ORB5000");

		}
		catch (exception ex) {
			throw ex;
		}
	}
	return true;
}
```

A.2 Get Matches Function

```
/***************Hybrid get matches function*******************************/
void GetMatches(const char* s, char* outStr, int outStrLen)
{
   std::string imagepath(s);
   /**********************Hu moments*********************************/
   vector<KeypointsWeightedResult> sampleArrMoments =
       HuMomentsHybridCalc(imagepath, knn);
   vector<KeypointsWeightedResult> sampleArrKeypoints =
       KeypointsHybridCalc(imagepath);

    /**********hybrid results object ******************/
   vector<KeypointsWeightedResult> sampleArrCombined = {
    {1,0,0},{2,0,0},{3,0,0},{4,0,0},{5,0,0},{6,0,0},{7,0,0},{8,0,0},
    {9,0,0},{10,0,0},{11,0,0},{12,0,0},{13,0,0},{14,0,0},{15,0,0},{16,0,0},
    {17,0,0},{18,0,0},{19,0,0},{20,0,0},{21,0,0},{22,0,0},{23,0,0},{24,0,0}};

   /*********Hybrid custom matching algorithm *********/
   for (int x = 0; x < sampleArrKeypoints.size(); x++) {
       int sampleId = sampleArrKeypoints[x].SampleId;
       sampleArrCombined[sampleId - 1].AvgDistance =
       (sampleArrKeypoints[x].AvgDistance == INT_MAX ? 0 :
       sampleArrCombined.size() / (x + 1)); //divided by species #
       sampleArrCombined[sampleId - 1].NumberOfOccurences =
       sampleArrKeypoints[x].NumberOfOccurences;
       for (int y = 0; y < sampleArrMoments.size(); y++) {
         if (sampleArrMoments[y].SampleId == sampleId) {
             sampleArrCombined[sampleId - 1].AvgDistance +=
            (sampleArrMoments[x].AvgDistance == INT_MAX ? 0
             : (y == 0 ? (sampleArrCombined.size() / (y + 1)) * 0.7 :
            sampleArrCombined.size() / (y + 1))); //Hu moments first match is
            poor so if y=0 weight result by 0.7 else no weight as subsequent
            match is >= keypoint match
```

```
            sampleArrCombined[sampleId - 1].NumberOfOccurences +=
             sampleArrMoments[x].NumberOfOccurences;
             continue;
             }
         }
    }
    std::sort(sampleArrCombined.begin(), sampleArrCombined.end(),
    CompareWeightValuesHybrid);
    //write to file
    std::string returnVal;
    for (int x = 0; x < sampleArrCombined.size(); x++) {
       returnVal += to_string(sampleArrCombined[x].SampleId) + "," +
       to_string(sampleArrCombined[x].NumberOfOccurences) + "," +
       to_string(sampleArrCombined[x].AvgDistance) + "|";
    }
    const char* c = returnVal.c_str();
    strcpy_s(outStr, outStrLen, c);
}
```

A.3 ORB5000 Keypoint RAM Classifier Train

```
/*************Keypoint classifier in memory train function***************/
void   train_keypoint_classifier_inmemory(std::string   train_path,   std::string
descriptorType) {


    bool isBinary = false;
    if (descriptorType == "ORB" || descriptorType == "ORB1000" ||
    descriptorType   ==   "ORB3000"   ||   descriptorType   ==   "ORB5000"   ||
    descriptorType == "ORB10000" || descriptorType == "AKAZE") {
             isBinary = true;
    }
    //this needs a Mat/Array per sample ID i.e one array for Sample 1 etc
    std::vector<cv::Mat> trainVector;
    //loop through all training images, extract features and populate data
    matrix and response vector
    //loop through species lables 'i'.
    for (int i = 1; i <= 24; i++) {
       std::string path = train_path + std::to_string(i) +
       "/training/training full sized";
       cv::Mat sampleDescriptors;
       for (const auto& entry : fs::directory_iterator(path)) {
       //extract features
       Mat img = imread(entry.path().string(), IMREAD_COLOR);
       cv::Mat descriptor = GetImageDescriptor(img, descriptorType, true);
                    sampleDescriptors.push_back(descriptor);
       }
       trainVector.push_back(sampleDescriptors);
       sampleDescriptors.release();

     }


       matcher                    =                    isBinary                    ?
       cv::FlannBasedMatcher(cv::makePtr<cv::flann::LshIndexParams>(10, 30, 0))
       : cv::FlannBasedMatcher(cv::makePtr<cv::flann::KDTreeIndexParams>(1));
       matcher.add(trainVector);
       matcher.train();

}
```

A.4 Hu-Moments Match Function

```
/***************Hu moments match vector calculation function************/
std::vector<KeypointsWeightedResult>    HuMomentsHybridCalc(std::string    path,
cv::Ptr<KNearest>& knn)
{
    vector<KeypointsWeightedResult> sampleArrMoments = {{1,0,0},{2,0,0},
    {3,0,0},{4,0,0},{5,0,0},{6,0,0},{7,0,0},{8,0,0},{9,0,0},{10,0,0},
    {11,0,0},{12,0,0},{13,0,0},{14,0,0},{15,0,0} ,{16,0,0},{17,0,0},
    {18,0,0},{19,0,0},{20,0,0},{21,0,0},{22,0,0},{23,0,0},{24,0,0}};
    Mat img = imread(path, IMREAD_GRAYSCALE);
    knn_result result = findNearestMoments(img, knn);
    //Loop through the found nearest neihbours
    for (int i = 0; i < result.neihbours.size(); i++) {
       int _neighbour = result.neihbours[i];
       //weighted array
       sampleArrMoments[_neighbour - 1].TotalDistance += result.distances[i];
              ++sampleArrMoments[_neighbour - 1].NumberOfOccurences;
    }
    /*******Hu moments custom matching algorithm***********/
    for (int x = 0; x < sampleArrMoments.size(); x++) {
       if (sampleArrMoments[x].NumberOfOccurences != 0)
         sampleArrMoments[x].AvgDistance = sampleArrMoments[x].TotalDistance /
         (sampleArrMoments[x].NumberOfOccurences *
          sampleArrMoments[x].NumberOfOccurences *
          sampleArrMoments[x].NumberOfOccurences);
       else
          sampleArrMoments[x].AvgDistance = INT_MAX;
     }
     std::sort(sampleArrMoments.begin(), sampleArrMoments.end(),
     CompareWeightValuesKnn);
     return sampleArrMoments;
}
```

A.5 ORB5000 Match function

```
/***************ORB5000 match vector calculation function************/
std::vector<KeypointsWeightedResult> KeypointsHybridCalc(std::string path)
{
   int minVal = 1; //for ORb5000
   vector<KeypointsWeightedResult> sampleArrKeypoints = {{1,0,0},{2,0,0},
   {3,0,0},{4,0,0},{5,0,0},{6,0,0},{7,0,0},{8,0,0},{9,0,0},{10,0,0},
   {11,0,0},{12,0,0},{13,0,0},{14,0,0},{15,0,0},{16,0,0},{17,0,0},
   {18,0,0},{19,0,0},{20,0,0},{21,0,0},{22,0,0},{23,0,0},{24,0,0}};
   Mat imgColour = imread(path, IMREAD_COLOR);

   cv::Mat query = GetImageDescriptor(imgColour, "ORB5000", true);
   std::vector<std::vector<cv::DMatch>> matches;
   matcher.knnMatch(query, matches, 2);

   std::vector<cv::DMatch> goodMatches;
   // Second neighbor ratio test.
   for (unsigned int i = 0; i < matches.size(); ++i) {
       if (matches[i].size() >= 2)
          if (matches[i][0].distance < matches[i][1].distance * 0.6)
              goodMatches.push_back(matches[i][0]);
   }
   for (int k = 0; k < goodMatches.size(); k++) {
       //weighted array
       sampleArrKeypoints[goodMatches[k].imgIdx].TotalDistance +=
```

```
            goodMatches[k].distance;
            ++sampleArrKeypoints[goodMatches[k].imgIdx].NumberOfOccurences;
        }
        //custom keypoint SI index
        for (int x = 0; x < sampleArrKeypoints.size(); x++) {
            if (sampleArrKeypoints[x].NumberOfOccurences != 0)
                sampleArrKeypoints[x].AvgDistance =
                (sampleArrKeypoints[x].TotalDistance == 0 ? minVal :
                sampleArrKeypoints[x].TotalDistance) /
                (sampleArrKeypoints[x].NumberOfOccurences *
                 sampleArrKeypoints[x].NumberOfOccurences *
                 sampleArrKeypoints[x].NumberOfOccurences);
             else
                 sampleArrKeypoints[x].AvgDistance = INT_MAX;
         }
         std::sort(sampleArrKeypoints.begin(), sampleArrKeypoints.end(),
         CompareWeightValuesKnn);

         return sampleArrKeypoints;
}
```

Appendix B Effect of Scaling on Matching

B.1 Scaling On Keypoint Matching

```
/*********Function used to obtain keypoint scaling results chapter 5*********/
int main(int argc, char** argv)
{
   string name;
   string descriptor;  //"SIFT, SURF, ORB, AKAZE, DAISY";
   std::cout << "Please enter descriptor..." << "\n";
   getline(cin, descriptor);

   for (int i = 1; i <= 6;i++) {
       name = "stem sample " + std::to_string(i); //either stems or leaves
       double factor = 0.1;
       Mat img_original = imread(path_to_sample_images + name + ".jpg",
       IMREAD_COLOR);
       if (img_original.empty())
       {
         std::cout << "File not found. Enter valid name" << "\n";
         continue;
       }

       std::ofstream myfile;
       myfile.open(descriptor + " Results.csv", std::ios::app);
       myfile << "\n" << name << "\n";
       myfile << "Keypoints 1," << "Keypoints 2," << "Matches," << "%
       Match," << "Factor, \n";

       while (factor<1) { //std::cin >> factor
           if (factor == 0)
              break;

          Mat img_resized;
          //resize image using declared factor
          cv::resize(img_original, img_resized, cv::Size(), factor, factor);
          //Run function to extract descriptors, compute matches and run 2nd
           neighbour test
           GetDescriptorsAndMatch(img_original, img_resized, descriptor, false,
           factor, name, myfile);
           waitKey(2);
           //std::cout << "\n" << "Please enter factor..." << "\n";

           factor += 0.1;
       }
    myfile.close();
    std::cout << "Please enter sample name..." << "\n";
  }
}
```

B.2 Scaling On Hu-Moments Matching

```
/*********Function used to obtain moments scaling results chapter 5*********/
int main(int argc, char** argv)
{

   for (int i = 1; i <= 6;i++) {
```

```
        double factor = 0.1;
        name = "stem sample " + std::to_string(i);
        Mat img_original = imread(path_to_sample_images + name + ".jpg" +
        IMREAD_COLOR);
        if (img_original.empty())
        {
           std::cout << "File not found. Enter valid name" << "\n";
           continue;
        }

        std::ofstream myfile;
        myfile.open("Hu Moments Stem Scaling Results.csv", std::ios::app);
        myfile << "Distance," << "Factor," << "Sample, \n";

        while (factor<1) {
            if (factor == 0)
                break;

          Mat img_resized;
          //resize image using declared factor
          cv::resize(img_original, img_resized, cv::Size(), factor, factor);

          Mat img_og_grey,img_res_grey;
          cv::cvtColor(img_original, img_og_grey, cv::COLOR_BGR2GRAY);
          cv::cvtColor(img_resized, img_res_grey, cv::COLOR_BGR2GRAY);

          //Run function to get hu moments
          std::vector<float> originalMoments = GetHuMoments(img_og_grey, true);
          std::vector<float> rotatedMoments = GetHuMoments(img_res_grey, true);
          //calculate Euclidean distance
          double distance = GetEucledianDistance(originalMoments,
                            rotatedMoments,7);

          myfile << distance << "," << factor << "," << i << endl;
          factor += 0.1;
        }
        myfile.close();
    }
}
```

Appendix C Effect of Rotation On Matching

C.1 Rotation On Keypoint Matching

```
/*********Function used to obtain keypoint rotation results chapter 5*********/
int main(int argc, char** argv)
{
   string descriptor;  //"SIFT, SURF, ORB, AKAZE, DAISY";
   std::cout << "Please enter descriptor..." << "\n";
   getline(cin, descriptor);
   for (int i = 1; i <= 5;i++) {
       name = "stem sample " + std::to_string(i);

       Mat img_original = imread(path_to_sample_images + name + ,IMREAD_COLOR);
       if (img_original.empty())
       {
           std::cout << "File not found. Enter valid name" << "\n";
           continue;
       }

       std::ofstream myfile;
       myfile.open(descriptor + " Leaves Rotation Results.csv", std::ios::app);
       myfile << "Keypoints 1," << "Keypoints 2," << "Matches," << "% Match,"
       << "Factor," << "Sample, \n";

       double angle = 0;
       while (angle<=360) {
          Mat img_rotated;
          //rotate image using declared factor

          cv::Point2f center((img_original.cols - 1) / 2.0, (img_original.rows
          - 1) / 2.0);
          cv::Mat rot = cv::getRotationMatrix2D(center, angle, 1.0);
          // determine bounding rectangle, center not relevant
          cv::Rect2f bbox = cv::RotatedRect(cv::Point2f(), img_original.size(),
          angle).boundingRect2f();
          // adjust transformation matrix
          rot.at<double>(0, 2) += bbox.width / 2.0 - img_original.cols / 2.0;
          rot.at<double>(1, 2) += bbox.height / 2.0 - img_original.rows / 2.0;
          cv::warpAffine(img_original, img_rotated, rot, bbox.size());


         //Run function to extract descriptors, compute matches and run 2nd
         neighbour test
         GetDescriptorsAndMatch(img_original, img_rotated, descriptor, false,
           angle, "stem sample ", i, myfile);
         waitKey(2);
         angle += 15;
       }
       myfile.close();
       std::cout << "Please enter sample name..." << "\n";
   }
}
```

C.1 Rotation on Hu-Moments

```
/*********Function used to obtain keypoint rotation results chapter 5*********/
int main(int argc, char** argv)
{
   string descriptor;  //"SIFT, SURF, ORB, AKAZE, DAISY";
   std::cout << "Please enter descriptor..." << "\n";
   getline(cin, descriptor);
   for (int i = 1; i <= 5;i++) {
       name = "stem sample " + std::to_string(i);

       Mat img_original = imread(path_to_sample_images + name + ,IMREAD_COLOR);
       if (img_original.empty())
       {
           std::cout << "File not found. Enter valid name" << "\n";
           continue;
       }

       std::ofstream myfile;
       myfile.open(descriptor + " Leaves Rotation Results.csv", std::ios::app);
       myfile << "Keypoints 1," << "Keypoints 2," << "Matches," << "% Match,"
       << "Factor," << "Sample, \n";

       double angle = 0;
       while (angle<=360) {
          Mat img_rotated;
          //rotate image using declared factor

          cv::Point2f center((img_original.cols - 1) / 2.0, (img_original.rows
          - 1) / 2.0);
          cv::Mat rot = cv::getRotationMatrix2D(center, angle, 1.0);
          // determine bounding rectangle, center not relevant
          cv::Rect2f bbox = cv::RotatedRect(cv::Point2f(), img_original.size(),
          angle).boundingRect2f();
          // adjust transformation matrix
          rot.at<double>(0, 2) += bbox.width / 2.0 - img_original.cols / 2.0;
          rot.at<double>(1, 2) += bbox.height / 2.0 - img_original.rows / 2.0;
          cv::warpAffine(img_original, img_rotated, rot, bbox.size());


          Mat img_og_grey,img_rot_grey;
          cv::cvtColor(img_original, img_og_grey, cv::COLOR_BGR2GRAY);
          cv::cvtColor(img_rotated, img_rot_grey, cv::COLOR_BGR2GRAY);
          //Run function to get hu moments
          std::vector<float> originalMoments = GetHuMoments(img_og_grey, true);
          std::vector<float> rotatedMoments = GetHuMoments(img_rot_grey, true);
          //calculate Euclidean distance
          double distance = GetEucledianDistance(originalMoments,rotatedMoments
          , 7);
          waitKey(2);
          angle += 15;
       }
       myfile.close();
       std::cout << "Please enter sample name..." << "\n";
   }
}
```

Appendix D Keypoint Get Descriptors And Match

```
//Function gets keypoint descriptors, calculates computational speed and saves
//results to CSV function
void GetDescriptorsAndMatch(Mat imgKnown, Mat imgQuery, string descriptorType,
bool brute_force, double factor, string sampleRoot, int sampleId, std::ofstream
&myfile) {
   // Detect the keyPoints and compute its descriptors.
   std::vector<cv::KeyPoint> keyPoints1, keyPoints2;
   cv::Mat descriptors1, descriptors2;
   bool binary_descriptor = false;
   double key1CalcTime, key2CalcTime;
   if (descriptorType == "ORB") {
   //***************orb descriptors***********************//
     binary_descriptor = true;
     cv::Ptr<cv::ORB> detector = cv::ORB::create();
     tick.reset(); tick.start();
     detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
     tick.stop();
     key1CalcTime = tick.getTimeMilli();
     tick.reset(); tick.start();
     detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
     tick.stop();
     key2CalcTime = tick.getTimeMilli();
    }
   else if (descriptorType == "ORB1000") {
    //***************orb1000 descriptors***********************//
     binary_descriptor = true;
     cv::Ptr<cv::ORB> detector = cv::ORB::create(1000);
     tick.reset(); tick.start();
     detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
     tick.stop();
     key1CalcTime = tick.getTimeMilli();
     tick.reset(); tick.start();
     detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
     tick.stop();
     key2CalcTime = tick.getTimeMilli();
    }
   else if (descriptorType == "ORB10000") {
    //***************orb10000 descriptors***********************//
     binary_descriptor = true;
     cv::Ptr<cv::ORB> detector = cv::ORB::create(10000);
     tick.reset(); tick.start();
     detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
     tick.stop();
     key1CalcTime = tick.getTimeMilli();
     tick.reset(); tick.start();
     detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
     tick.stop();
     key2CalcTime = tick.getTimeMilli();
    }
   else if (descriptorType == "AKAZE") {
    //***************AKAZE descriptors***********************//
     binary_descriptor = true;
     cv::Ptr<cv::AKAZE> detector = cv::AKAZE::create();
     tick.reset(); tick.start();
     detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
     tick.stop();
     key1CalcTime = tick.getTimeMilli();
     tick.reset(); tick.start();
```

```
    detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
    tick.stop();
    key2CalcTime = tick.getTimeMilli();
  }
 else if (descriptorType == "SIFT") {
    //***************sift descriptors*******************//
   cv::Ptr <cv::xfeatures2d::SIFT> detector = cv::xfeatures2d::SIFT::create();
   //use minhessian btwn 300 and 500
   tick.reset(); tick.start();
   detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
   tick.stop();
   key1CalcTime = tick.getTimeMilli();
   tick.reset(); tick.start();
   detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
   tick.stop();
   key2CalcTime = tick.getTimeMilli();
  }
 else if (descriptorType == "SURF") {
   //***************surf descriptors*******************//
   cv::Ptr <cv::xfeatures2d::SURF> detector =
           cv::xfeatures2d::SURF::create(300); //In actual cases, it is better
           to have a value 300-500
   tick.reset(); tick.start();
   detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
   tick.stop();
   key1CalcTime = tick.getTimeMilli();
   tick.reset(); tick.start();
   detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
   tick.stop();
   key2CalcTime = tick.getTimeMilli();
  }
  else if (descriptorType == "KAZE") {
  //***************KAZE descriptors*******************//
   cv::Ptr<cv::Feature2D> detector = cv::KAZE::create();
   tick.reset(); tick.start();
   detector->detectAndCompute(imgKnown, cv::Mat(), keyPoints1, descriptors1);
   tick.stop();
   key1CalcTime = tick.getTimeMilli();
   tick.reset(); tick.start();
   detector->detectAndCompute(imgQuery, cv::Mat(), keyPoints2, descriptors2);
   tick.stop();
   key2CalcTime = tick.getTimeMilli();
  }
  //daisy descriptor is a dense descriptor.
  else if (descriptorType == "DAISY") {
      //***************Daisy descriptors*******************//
    cv::Ptr <cv::xfeatures2d::SURF> detector =cv::xfeatures2d::SURF::create();
    cv::Ptr <cv::xfeatures2d::DAISY> extractor =
            cv::xfeatures2d::DAISY::create();
    tick.reset(); tick.start();
    detector->detect(imgKnown, keyPoints1);
    extractor->compute(imgKnown, keyPoints1, descriptors1);
    tick.stop();
    key1CalcTime = tick.getTimeMilli();
    tick.reset(); tick.start();
    detector->detect(imgQuery, keyPoints2);
    extractor->compute(imgQuery, keyPoints2, descriptors2);
    tick.stop();
    key2CalcTime = tick.getTimeMilli();
}
```

```
//cout keypoint size
std::cout << keyPoints1.size() << " Keypoints in 1" << " --- ";
std::cout << keyPoints2.size() << " Keypoints in 2 --- ";
if (keyPoints1.size() > 2 && keyPoints2.size()>2) { //both arrays must have more
than 2 matches othrwise it is impossible to run the 2nd neighbour ratio test
    std::vector<cv::DMatch> matches;
    Match(descriptors1, descriptors2, matches, brute_force, binary_descriptor,
    factor);
    //pick the one with the smaller keypoints. find the percentage of matches
    versus keypoints
    double match_denominator, keypoint_denominator;
    if (keyPoints2.size() > keyPoints1.size()) {
       match_denominator = keyPoints1.size();
       keypoint_denominator = keyPoints2.size();
    }
    else {
       match_denominator = keyPoints2.size();
       keypoint_denominator = keyPoints1.size();
    }
    double match_numerator = matches.size();

   //****used for CSV saving image to image percentage matching and variance
   segregated image to image
   //SaveKeypointVarianceResults(descriptorType, keyPoints1, keyPoints2,
   //matches, sampleId, myfile);

   //****used for CSV saving scaling and rotation
   //SaveResults(descriptorType, keyPoints1, keyPoints2, matches, factor,
   //sampleId, myfile);

   //****used for CSV saving computational speed
   //SaveComputationalSpeedResults(descriptorType, keyPoints1,
   //keyPoints2,matches, key1CalcTime, key2CalcTime, myfile);

   //****used for CSV saving nearest neighbour results *****/
   //myfile << keyPoints1.size() << "," << keyPoints2.size() << "," <<
   //matches.size() << "," << (match_numerator / match_denominator) * 100 <<
   //"," << factor << ",\n";

   /*****************uncomment to draw keypoints separately*/
   //cv::Mat image_keypoints1;
   //cv::drawKeypoints(imgKnown, keyPoints1, image_keypoints1);
   //namedWindow("Keypoints1", WINDOW_KEEPRATIO);
   //cv::imshow("Keypoints1", image_keypoints1);

   /**** uncomment to Draw matches.*/
   //cv::Mat image_matches;
   //cv::drawMatches(imgKnown, keyPoints1, imgQuery, keyPoints2, matches,
     image_matches);
   //namedWindow("Matches", WINDOW_KEEPRATIO);
   //cv::imshow("Matches", image_matches);

   /*Uncomment below to save match image to file for rotation and scaling*/
   //imwrite(path_to_save, image_matches);
   }
}
```

Appendix E ORB5000 Scaling and Rotation Images

E.1 ORB5000 Leaves Scaling

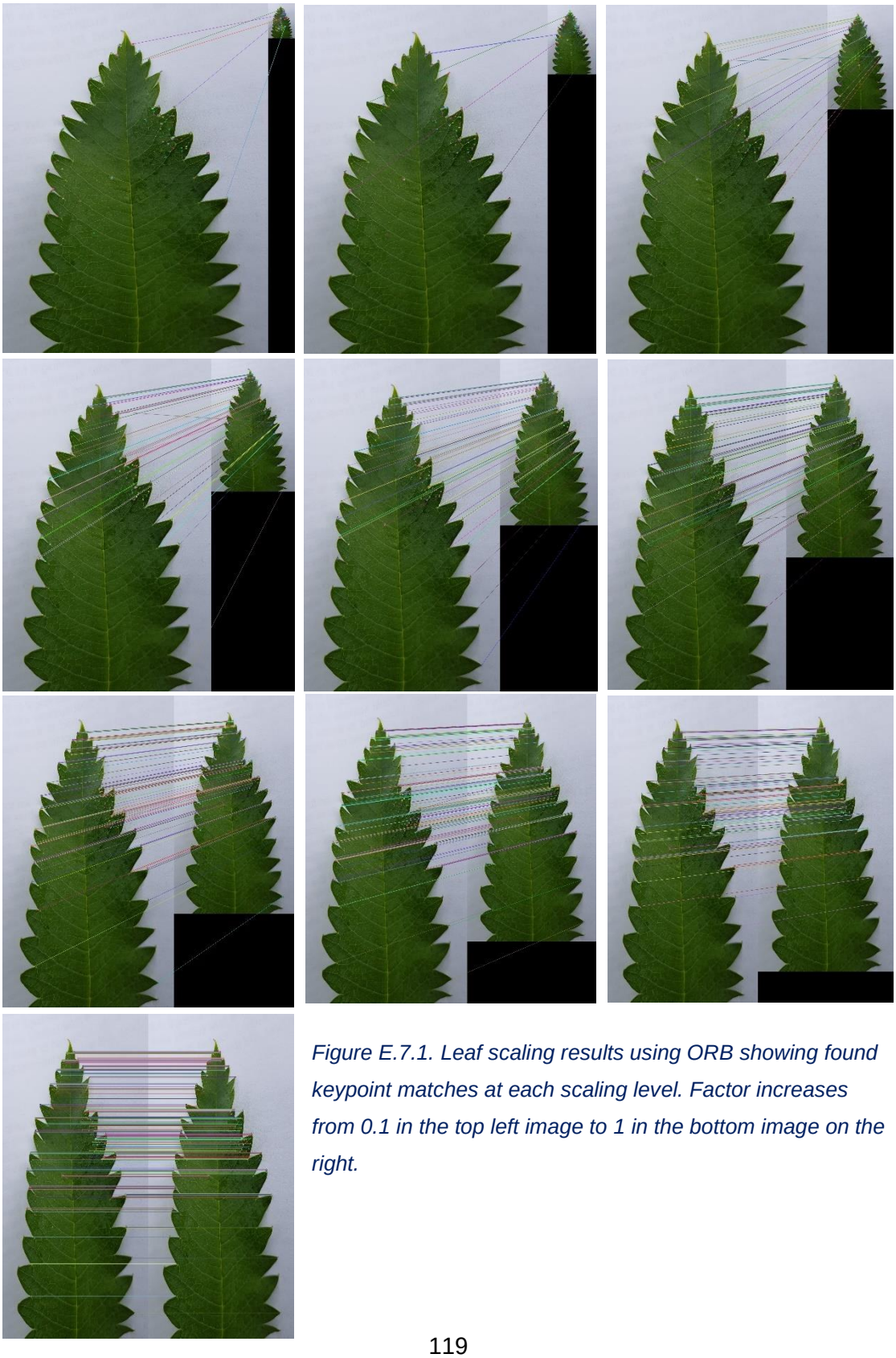


Figure E.7.1. Leaf scaling results using ORB showing found keypoint matches at each scaling level. Factor increases from 0.1 in the top left image to 1 in the bottom image on the right.

E.2 ORB5000 Stems Scaling

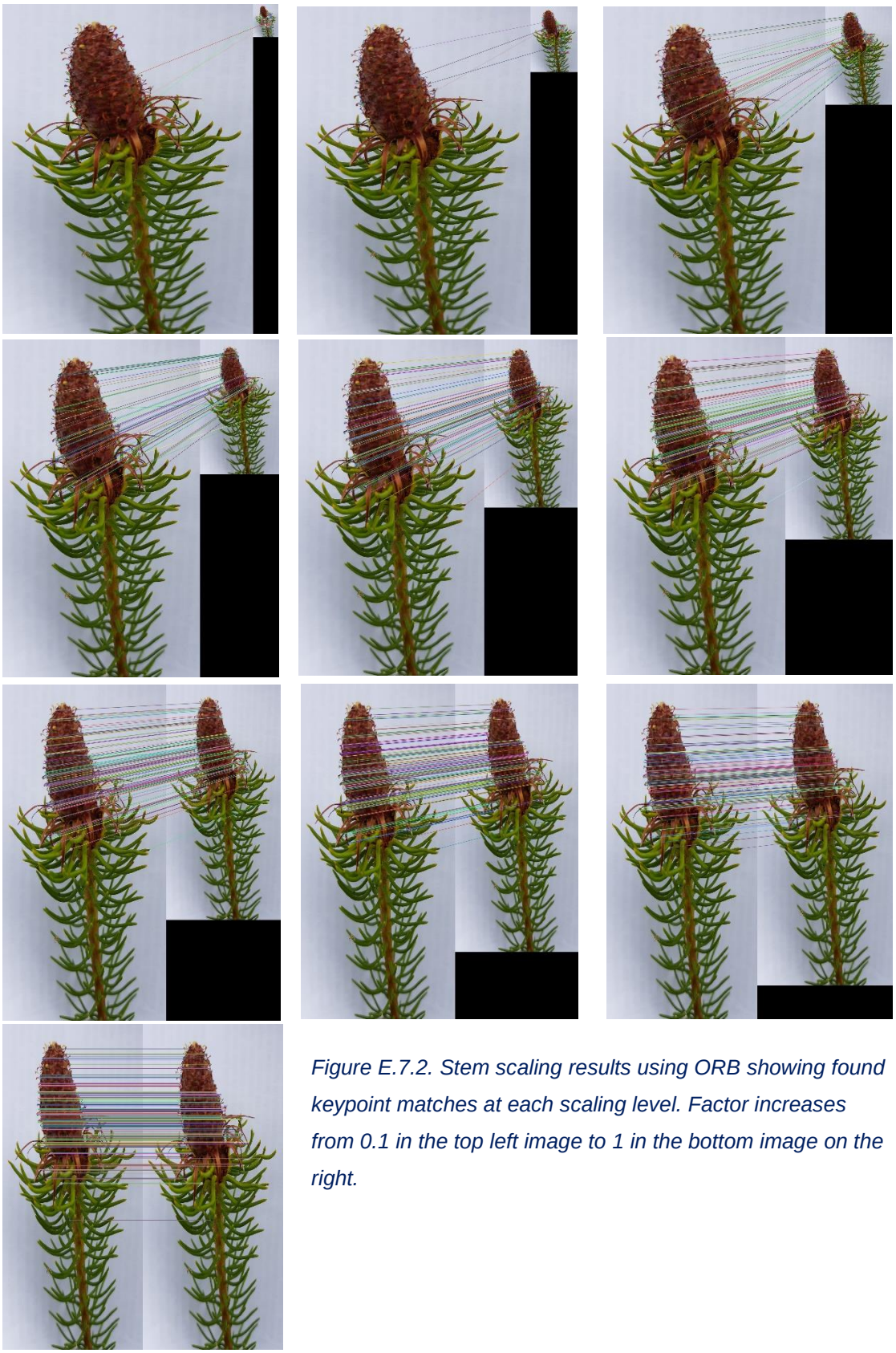
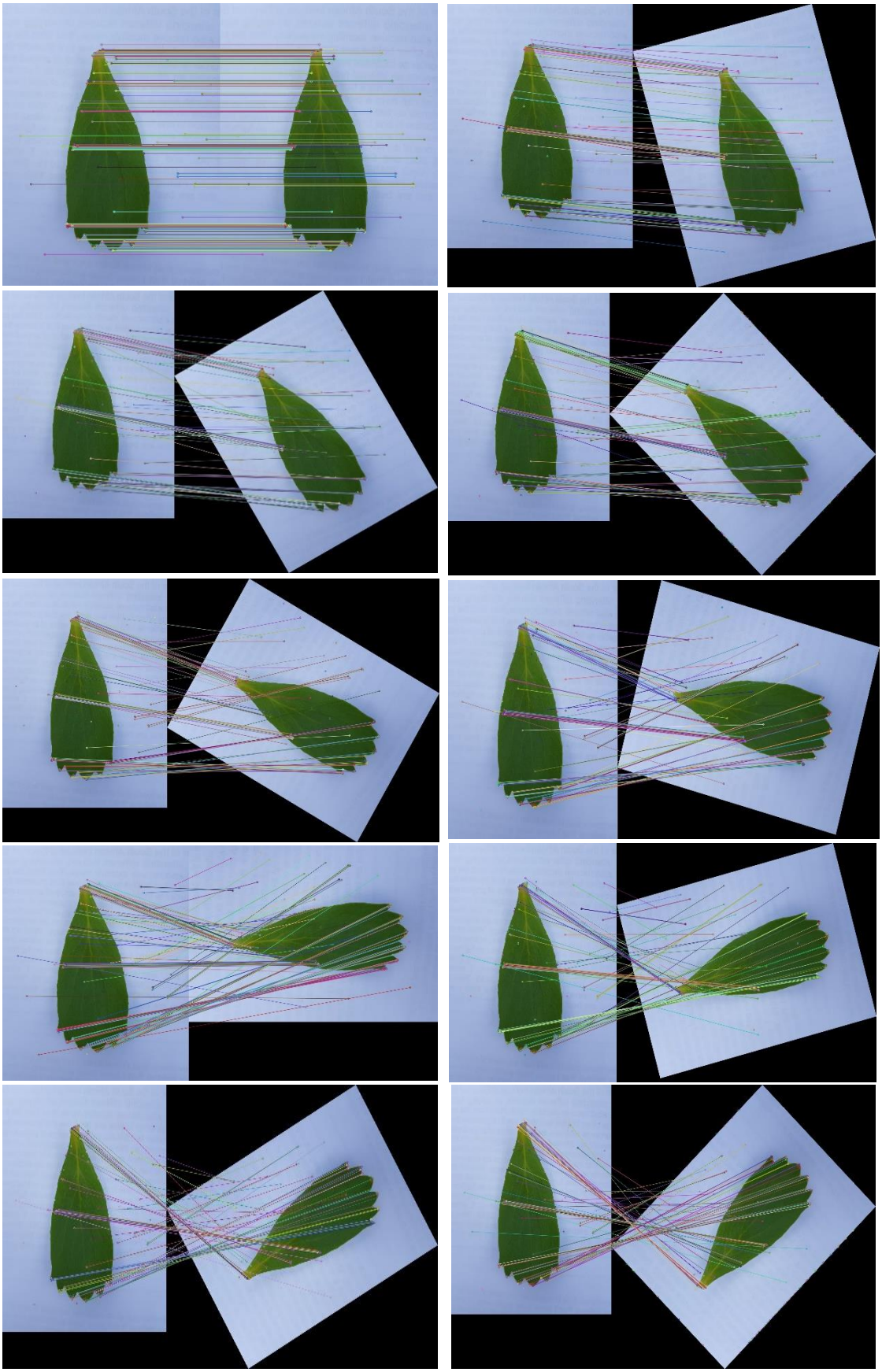


Figure E.7.2. Stem scaling results using ORB showing found keypoint matches at each scaling level. Factor increases from 0.1 in the top left image to 1 in the bottom image on the right.

E.3 ORB5000 Leaves Rotation



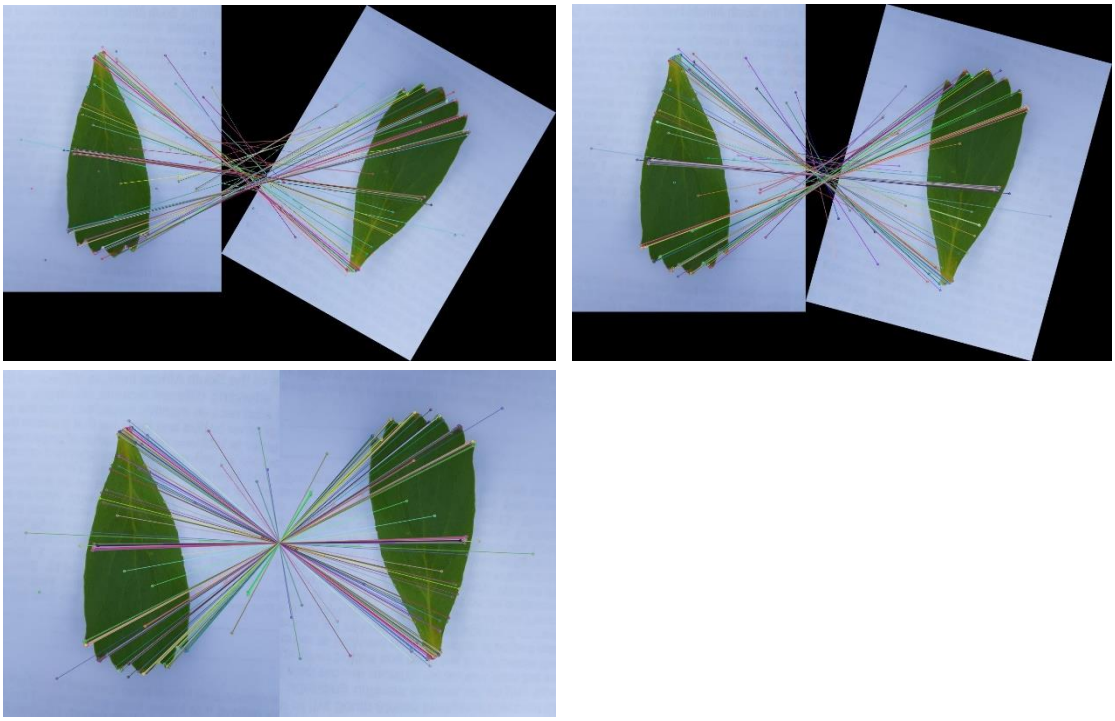


Figure E.7.3. Leaf rotation results using ORB showing keypoint matches at each rotation angle starting from 0 in the top right corner to 180 degrees in the bottom image on the right.

E.4 ORB5000 Stems Rotation

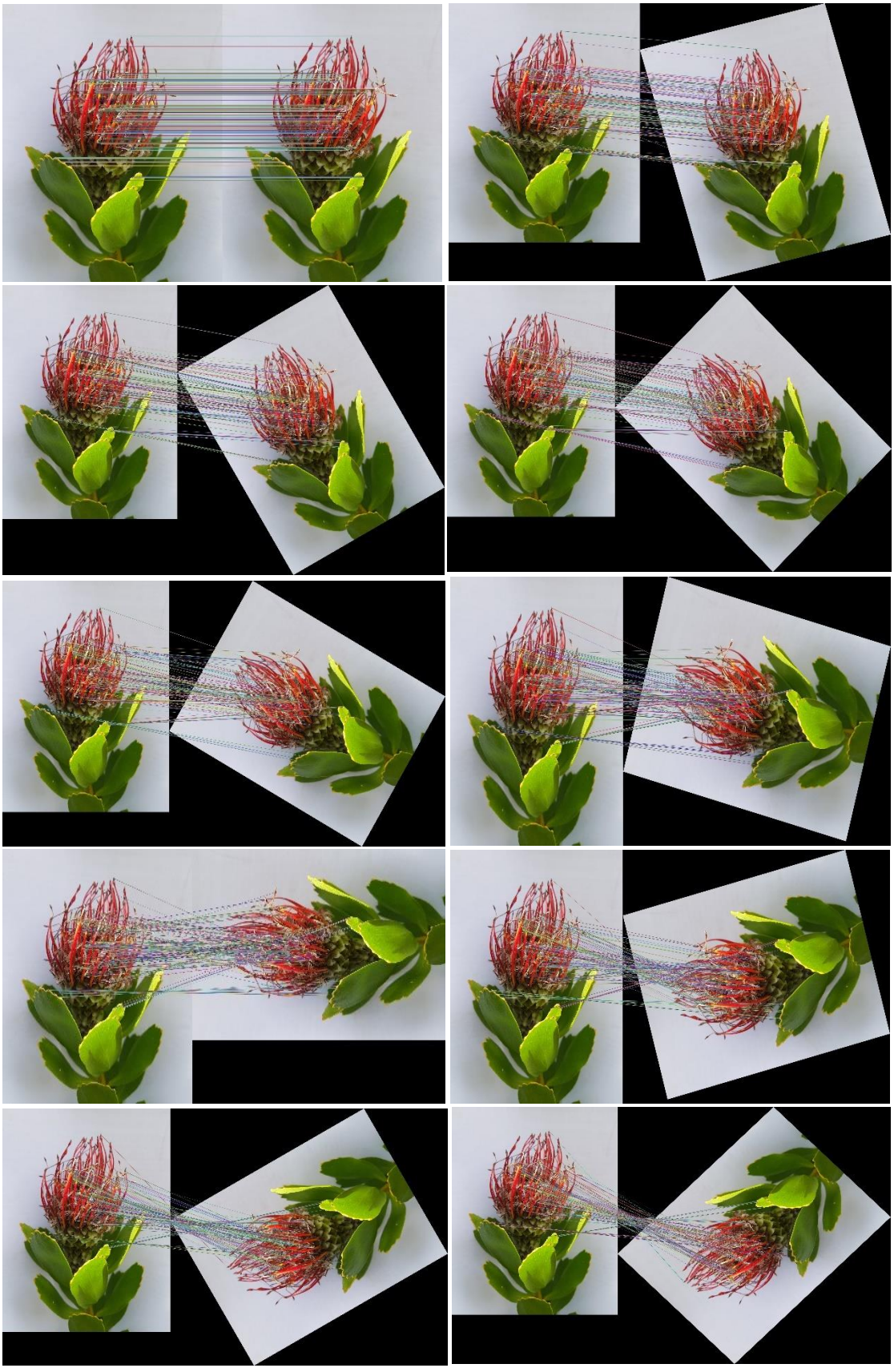


Figure E.7.4. Stem rotation results using ORB showing keypoint matches at each rotation angle starting from 0 in the top right corner to 180 degrees in the bottom image on the right.

Appendix F FLANN Matcher Parameter Analysis

F.1 SIFT KDTreeIndexParams Analysis

Table F.7.1. A snippet of 30 of the 103 SIFT test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. KDTreeIndexParams(10) showed a high degree of data loss with only 30% of test samples matched.

SIFT KDTreeIndexParams (10) - Load Time: 320.9s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	3371.14	2	FALSE	FALSE	FALSE	11	1	96.7781	7	1	133.173	0	0	2.15E+09
1	3165.36	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	2471.98	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	2003.43	1	TRUE	TRUE	TRUE	2	1	93.3167	0	0	2.15E+09	0	0	2.15E+09
2	1915.67	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	1483.08	1	FALSE	FALSE	FALSE	9	1	58.3095	0	0	2.15E+09	0	0	2.15E+09
3	3611.7	2	TRUE	TRUE	TRUE	3	2	34.2391	0	0	2.15E+09	0	0	2.15E+09
3	2391.19	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	3090.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	2359.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	4655.51	1	FALSE	FALSE	FALSE	8	1	198.935	0	0	2.15E+09	0	0	2.15E+09
4	6190.84	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	3867.76	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	3993.21	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	2896.6	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	4964.8	89	TRUE	TRUE	TRUE	5	89	0.0158638	0	0	2.15E+09	0	0	2.15E+09
5	4285.45	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	1944.57	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	370.344	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	355.284	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	642.966	4	FALSE	FALSE	FALSE	4	4	2.91186	0	0	2.15E+09	0	0	2.15E+09
6	245.448	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	3657.78	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	1275.86	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	3060.49	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	3981.57	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	2377.26	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	3398.17	2	TRUE	TRUE	TRUE	9	1	55.4076	12	1	81.4801	0	0	2.15E+09
9	2205.64	1	FALSE	FALSE	FALSE	5	1	158.076	0	0	2.15E+09	0	0	2.15E+09
9	2283.87	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09

Table F.7.2. A snippet of 30 of the 103 SIFT test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. KDTreeIndexParams(3) performed better than KDTreeIndexParams(10) although still showed a high degree of data loss with only 39% of test samples matched.

SIFT KDTreeIndexParams (3) - Load Time: 281.9s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	3130.8	1	FALSE	FALSE	FALSE	11	1	96.7781	0	0	2.15E+09	0	0	2.15E+09
1	2950.5	1	FALSE	FALSE	FALSE	10	1	146.922	0	0	2.15E+09	0	0	2.15E+09
1	2425.68	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	1816.38	2	TRUE	TRUE	TRUE	2	1	93.3167	7	1	136.077	0	0	2.15E+09
2	1791.79	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	1424.16	1	FALSE	FALSE	FALSE	9	1	58.3095	0	0	2.15E+09	0	0	2.15E+09
3	3395.58	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	2289.31	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	2988.05	1	FALSE	FALSE	FALSE	18	1	139.621	0	0	2.15E+09	0	0	2.15E+09
4	2300.58	1	TRUE	TRUE	TRUE	4	1	157.08	0	0	2.15E+09	0	0	2.15E+09
4	4444.99	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	5588.57	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	3556.48	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	3594.58	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	2812.45	1	TRUE	TRUE	TRUE	5	1	157.445	0	0	2.15E+09	0	0	2.15E+09
5	4519.06	88	TRUE	TRUE	TRUE	5	87	1.43317	13	1	106.348	0	0	2.15E+09
5	4387.94	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	2034.93	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	293.02	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	382.437	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	657.783	4	FALSE	FALSE	FALSE	4	4	11.6474	0	0	2.15E+09	0	0	2.15E+09
6	285.596	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	3626.26	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	1310.87	1	FALSE	FALSE	FALSE	21	1	146.86	0	0	2.15E+09	0	0	2.15E+09
8	2989.56	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	4036.89	2	TRUE	TRUE	TRUE	8	1	82.662	21	1	140.957	0	0	2.15E+09
8	2374.55	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	3290.7	2	TRUE	TRUE	TRUE	9	1	55.4076	12	1	81.4801	0	0	2.15E+09
9	2192.44	2	FALSE	FALSE	FALSE	20	1	39.8372	5	1	158.076	0	0	2.15E+09
9	2112.06	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09

F.2 SURF KDTreeIndexParams Analysis

Table F.7.3. A snippet of 30 of the 103 SURF test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. KDTreeIndexParams(10) matched 49% of test images to a sample.

SURF KDTreeIndexParams (10) - Load Time: 330.7s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	5634.97	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	5112.32	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	3995.8	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	4043.94	1	FALSE	FALSE	FALSE	16	1	0.0895272	0	0	2.15E+09	0	0	2.15E+09
2	3170.2	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	3127.87	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	4749.42	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	2964.36	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	6315.08	1	TRUE	TRUE	TRUE	4	1	0.157924	0	0	2.15E+09	0	0	2.15E+09
4	5291	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	4839.45	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	13273.4	2	TRUE	TRUE	TRUE	4	2	0.0221083	0	0	2.15E+09	0	0	2.15E+09
4	4947	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	5564.98	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	4725.66	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	7829.38	79	TRUE	TRUE	TRUE	5	79	1.72E-05	0	0	2.15E+09	0	0	2.15E+09
5	6338.49	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	4158.19	1	TRUE	TRUE	TRUE	5	1	0.0700451	0	0	2.15E+09	0	0	2.15E+09
6	5793.81	3	TRUE	TRUE	TRUE	6	3	0.00246085	0	0	2.15E+09	0	0	2.15E+09
6	3145.69	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	1782.41	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	3719.08	1	FALSE	FALSE	FALSE	7	1	0.0172872	0	0	2.15E+09	0	0	2.15E+09
7	6190.15	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	3685.44	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	5871.12	2	TRUE	TRUE	TRUE	8	2	0.0201464	0	0	2.15E+09	0	0	2.15E+09
8	6901.03	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	4563.85	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	6394.62	3	FALSE	TRUE	TRUE	8	1	0.0699505	22	1	0.0786228	9	1	0.115571
9	4758	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	5051.13	1	TRUE	TRUE	TRUE	9	1	0.0460521	0	0	2.15E+09	0	0	2.15E+09

Table F.7.4. A snippet of 30 of the 103 SURF test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. KDTreeIndexParams(3) matched 55% of test images to a sample.

SURF KDTreeIndexParams (3) - Load Time: 283.1s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	5210.93	2	FALSE	FALSE	FALSE	2	1	0.0650244	5	1	0.0785376	0	0	2.15E+09
1	4896.95	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
1	3887.11	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	3964.22	1	FALSE	FALSE	FALSE	16	1	0.0895272	0	0	2.15E+09	0	0	2.15E+09
2	3018.82	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
2	2864.97	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
3	4474.94	1	FALSE	FALSE	FALSE	2	1	0.0630447	0	0	2.15E+09	0	0	2.15E+09
3	2714.31	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	5908	1	FALSE	FALSE	FALSE	9	1	0.11945	0	0	2.15E+09	0	0	2.15E+09
4	5262.66	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	5045.53	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
4	12197.2	1	TRUE	TRUE	TRUE	4	1	0.042152	0	0	2.15E+09	0	0	2.15E+09
4	4915.99	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	5066.64	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	4642.22	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
5	7488.14	89	TRUE	TRUE	TRUE	5	88	0.00122249	13	1	0.141144	0	0	2.15E+09
5	6054.74	1	FALSE	FALSE	FALSE	14	1	0.180583	0	0	2.15E+09	0	0	2.15E+09
5	3775.49	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	5289.73	2	TRUE	TRUE	TRUE	6	2	0.00685487	0	0	2.15E+09	0	0	2.15E+09
6	2943.57	1	TRUE	TRUE	TRUE	6	1	0.200676	0	0	2.15E+09	0	0	2.15E+09
6	1764.94	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
6	3680.53	1	TRUE	TRUE	TRUE	6	1	0.0160098	0	0	2.15E+09	0	0	2.15E+09
7	6659.04	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
7	3756.01	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	5403.72	2	TRUE	TRUE	TRUE	8	2	0.0402928	0	0	2.15E+09	0	0	2.15E+09
8	6179.36	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
8	4458.53	2	TRUE	TRUE	TRUE	8	2	0.0273476	0	0	2.15E+09	0	0	2.15E+09
9	5916.97	3	FALSE	TRUE	TRUE	8	1	0.0699505	22	1	0.0786228	9	1	0.115571
9	4577.63	0	FALSE	FALSE	FALSE	0	0	2.15E+09	0	0	2.15E+09	0	0	2.15E+09
9	4702.17	1	TRUE	TRUE	TRUE	9	1	0.0460521	0	0	2.15E+09	0	0	2.15E+09

F.3 ORB5000 LshIndex Analysis

Table F.7.5. A snippet of 30 of the 103 ORB5000 test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. LshIndex(12,20,0) matched 86% of test images to a sample and of these 38% were correct.

ORB5000 LshIndex (12,20,0)- Load Time: 283.1s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
2	1674.87	2	TRUE	TRUE	TRUE	2	1	14	1	1	19	0	0	2
4	3502.63	12	TRUE	TRUE	TRUE	4	7	0.134111	10	1	3	15	1	4
4	3590.63	1	TRUE	TRUE	TRUE	4	1	22	0	0	2.15E+09	0	0	4
5	2119.56	239	TRUE	TRUE	TRUE	5	237	0.0003916	3	1	5	1	1	5
5	2231.73	3	TRUE	TRUE	TRUE	5	1	10	23	1	11	14	1	5
6	2450.68	2	TRUE	TRUE	TRUE	6	2	1	0	0	2.15E+09	0	0	6
6	1000.75	1	TRUE	TRUE	TRUE	6	1	3	0	0	2.15E+09	0	0	6
8	2862.04	11	TRUE	TRUE	TRUE	8	5	0.232	7	1	1	9	1	8
8	2657.01	8	TRUE	TRUE	TRUE	8	4	0.453125	6	1	2	24	1	8
9	3163.23	7	TRUE	TRUE	TRUE	9	2	0.125	7	1	4	12	1	9
9	2986.85	11	TRUE	TRUE	TRUE	9	4	0.1875	7	1	1	13	1	9
10	2458.19	4	TRUE	TRUE	TRUE	10	4	0.859375	0	0	2.15E+09	0	0	10
10	2204.27	2	TRUE	TRUE	TRUE	10	2	3.875	0	0	2.15E+09	0	0	10
10	2390.33	5	TRUE	TRUE	TRUE	10	4	1.01563	21	1	7	0	0	10
10	1656.72	26	TRUE	TRUE	TRUE	10	26	0.0221893	0	0	2.15E+09	0	0	10
10	1547.31	4	TRUE	TRUE	TRUE	10	4	0.734375	0	0	2.15E+09	0	0	10
10	1294.56	9	TRUE	TRUE	TRUE	10	8	0.152344	8	1	8	0	0	10
12	2011.28	70	TRUE	TRUE	TRUE	12	69	0.00425559	23	1	3	0	0	12
13	2659.75	2	TRUE	TRUE	TRUE	13	1	7	3	1	8	0	0	13
13	3016.8	9	TRUE	TRUE	TRUE	13	2	0.625	16	2	0.625	12	1	13
14	2267.82	93	TRUE	TRUE	TRUE	14	93	0.00269283	0	0	2.15E+09	0	0	14
14	2142.66	123	TRUE	TRUE	TRUE	14	123	0.0013714	0	0	2.15E+09	0	0	14
14	1861.38	1	TRUE	TRUE	TRUE	14	1	18	0	0	2.15E+09	0	0	14
15	2268.1	5	TRUE	TRUE	TRUE	15	2	1.5	7	2	1.5	23	1	15
15	2178.68	2	TRUE	TRUE	TRUE	15	1	6	17	1	7	0	0	15
15	1556.41	1	TRUE	TRUE	TRUE	15	1	18	0	0	2.15E+09	0	0	15
16	2198.13	23	TRUE	TRUE	TRUE	16	21	0.053126	13	1	1	7	1	16
17	2650.28	10	TRUE	TRUE	TRUE	17	6	0.296296	8	1	1	20	1	17
17	2509.02	4	TRUE	TRUE	TRUE	17	2	3	8	1	17	16	1	17
18	1935.83	142	TRUE	TRUE	TRUE	18	140	0.00114978	8	1	1	14	1	18

Table F.7.6. A snippet of 30 of the 103 ORB5000 test results obtained from running test images through the file k-NN classifier then extracting the similarity index in the custom matching algorithm. LshIndex (12,33,0) matched 98% of test images to a sample and of these 49% were correct.

ORB5000 LshIndex (12,33,0)- Load Time: 497.8.1s														
			Correct Match											
Sample ID	Calc Time (ms)	Number of Matches	No. 1	Top 5	Top 10	Match 1	Match 1 Occurrences	Match 1 SI	Match 2	Match 2 Occurrences	Match 2 SI	Match 3	Match 3 Occurrences	Match 3 SI
1	1376.21	20	FALSE	FALSE	FALSE	13	3	4.25926	9	3	5.11111	21	2	8.75
1	1406.46	24	TRUE	TRUE	TRUE	1	7	0.737609	3	3	3.07407	23	2	10
1	942.35	18	TRUE	TRUE	TRUE	1	2	4.5	18	3	5.55556	4	2	6.5
2	651.943	10	FALSE	TRUE	TRUE	9	1	5	18	2	5.125	23	2	8.125
2	533.025	13	TRUE	TRUE	TRUE	2	3	3.74074	11	1	6	18	2	8.75
2	499.84	8	FALSE	TRUE	TRUE	22	2	5.75	4	2	10.125	2	1	25
3	1079.24	28	TRUE	TRUE	TRUE	3	5	0.92	9	1	1	1	3	3.11111
3	992.754	26	TRUE	TRUE	TRUE	3	11	0.203606	12	2	3	4	2	7.5
4	1379.8	42	TRUE	TRUE	TRUE	4	12	0.146412	22	4	2.53125	10	1	3
4	1200.68	24	TRUE	TRUE	TRUE	4	5	1.008	7	4	1.28125	9	3	2.22222
4	1053.29	26	FALSE	FALSE	FALSE	16	3	3.74074	11	3	3.85185	9	1	4
4	1287.62	18	FALSE	TRUE	TRUE	2	2	6.125	4	2	6.25	17	2	8.875
4	942.886	16	FALSE	TRUE	TRUE	9	2	6.625	4	2	8.375	7	2	9
5	784.75	17	FALSE	FALSE	FALSE	23	2	6.625	7	1	7	10	1	9
5	735.231	13	FALSE	FALSE	FALSE	16	2	7.5	24	1	23	21	1	34
5	791.394	264	TRUE	TRUE	TRUE	5	251	0.000331431	11	3	4.44444	3	1	5
5	763.544	15	FALSE	TRUE	TRUE	23	2	7.5	8	2	8.25	21	2	9.875
5	544.643	11	FALSE	FALSE	FALSE	24	1	3	10	1	12	11	2	12.625
6	489.64	3	TRUE	TRUE	TRUE	6	2	1	15	1	29	0	0	2.15E+09
6	407.869	7	FALSE	TRUE	TRUE	4	1	3	6	1	4	22	3	4.18518
6	315.405	4	TRUE	TRUE	TRUE	6	2	5.25	11	1	46	23	1	66
6	336.352	1	TRUE	TRUE	TRUE	6	1	3	0	0	2.15E+09	0	0	2.15E+09
7	980.083	25	TRUE	TRUE	TRUE	7	7	0.752187	9	1	2	10	2	3
7	760.212	7	FALSE	FALSE	FALSE	13	1	2	17	1	2	8	2	6.625
8	1009.46	27	TRUE	TRUE	TRUE	8	7	0.218659	9	4	0.828125	21	1	2
8	869.767	20	TRUE	TRUE	TRUE	8	3	0.851852	22	1	1	11	3	4.14815
8	736.204	21	TRUE	TRUE	TRUE	8	5	0.536	6	1	2	13	4	2.5625
9	1159.92	16	TRUE	TRUE	TRUE	9	3	0.185185	4	1	2	12	2	3.25
9	1136.61	23	FALSE	TRUE	TRUE	24	2	0.25	8	5	0.888	9	5	1.056
9	1012.88	37	TRUE	TRUE	TRUE	9	9	0.130315	7	5	0.912	11	6	1.11111