Taking Shape
The Data Science of Elastic Shape Analysis with Practical Applications

Arianna Salili-James
Department of Mathematics
Brunel University London

Declaration

I declare that this thesis, submitted to Brunel University London, was composed by myself, that the contents contained herein is my own original work except where otherwise acknowledged in the text or by references, and that this work has not been submitted for any other degree in any other schools or places.

Abstract

A mathematical curve can represent many different objects, both physical and abstract, from the outline curve of an artefact in an image to the weight of growing animal to the set of frequencies used in a sound. Regardless of these variations, the curves can almost always vary non-linearly. One way to study shapes and their potential variations is elastic shape analysis, a rich theory of which has developed over the past twenty years. However, methods of elastic shape analysis are seldom utilized in practical applications on real-world data, especially outside of the mathematical shape analysis community.

Our aim in this thesis is to explore some practical applications of elastic shape analysis. To do this, we work with various types of shape data, the majority of which are based on image datasets. As our focus is on two-dimensional curves, it is important to be able to robustly extract contours from images, before we can apply elastic shape analysis tools. In order to analyse the shapes in a dataset, we turn to methods of machine learning, to investigate the applications of elastic shape analysis in classification.

In this thesis, we introduce an anthology of projects, in order to emphasise and understand the potential of elastic shape analysis in practical applications. There are four main projects in this thesis: (i) Classification of objects using outlines and the comparisons between methods of elastic shape analysis, geometric morphometrics, and human experts, with a focus on ancient Greek vases, (ii) Mussel species identification and a demonstration that shape may *not* be enough in some applications, (iii) A novel tool to monitor the development of kākāpō chicks, and (iv) Classifying individual kiwi based on acoustic data from their calls.

By combining tools from computer vision and machine learning with methods of elastic shape analysis, we introduce a practical framework for the application of elastic shape analysis, through a data science lens.

Acknowledgements

I would like to thank Stephen Marsland for being the best supervisor I could have asked for, despite being thousands of miles away. And I would like to thank Matthias Maischak for his continuous support at all times. I would also like to thank all the collaborators who were involved in the projects that make up this PhD, in particular to Armand Leroi, and to Andrew Digby. Furthermore, I would like to thank the PGR team at Brunel University London for their invaluable support, and Martins Bruveris for giving me the opportunity to begin this PhD in the first place.

I would like to express my deepest gratitude to my wonderful friends and family. From my closest friends who have supported me throughout my life, and put up with my absences during these last few years, to my fellow PhD colleagues who shared the journey with me, and motivated me to keep on going until the very end. In particular, thank you to my sister Aida, for always being there for me and constantly keeping me motivated. And to my Benjamin, I would not have been able to do this PhD without you. Thank you for always having my back through the good times and the difficult times, and thank you for your constant help, support, expertise, and kindness. To my parents, I am eternally grateful for everything you have done for me and continue to do for me. Thank you for lighting my love for mathematics and starting me on this journey.

I dedicate this thesis to my family.

Preface

This PhD and thesis have been composed by myself. But the projects within have been in collaboration with many other scientists from different areas. Collaborations are vital to science and without it, science cannot progress. To emphasise my appreciation, for the remainder of this thesis, I use "*we*" rather than "*I*".

Contents

List of Algorithms

"Must a name mean something?" Alice asked doubtfully. "Of course it must," Humpty Dumpty said with a short laugh: "MY name means the shape I am – and a good handsome shape it is, too. With a name like yours, you might be any shape, almost."

Lewis Caroll, Alice Through the Looking-Glass

Chapter 1

Introduction

Shape analysis is the mathematical study of shapes, abstractly represented as continuous curves. The mathematical spaces where shapes are described are non-linear, meaning that the data-based analysis of shape requires specialist tools. In this interdisciplinary data science thesis we focus on developing methods to apply the tools of shape analysis to real-world applications. We introduce these applications using different projects that form the individual chapters of this thesis. By delving into the areas of mathematical shape analysis, computer vision, and machine learning, amongst others, our aim is to develop novel tools and procedures to enable modern shape analysis methods to be applied by domain scientists, without requiring too much mathematical knowledge.

We begin this introduction by defining the crucial word that this research is based on: *shape*. Next, we will briefly outline the field of shape analysis (a more detailed overview is presented in the following chapter) and functional data analysis. And finally, we will detail the research projects that comprise this PhD, discuss their motivations and objectives; and outline the roadmap of this thesis.

1.1 What Is Shape?

We can learn a lot about an object from its physical attributes, such as its size and texture. However, the key information is often in the object's *shape*.

In everyday language, the word *shape* is used varyingly, whether it is to describe a three

dimensional surface, or the outline of an object. In defining *shape*, it can be helpful to distinguish between *shape* and *form*, although this is rarely done. Even the Oxford dictionary lists one definition of "shape" as "form", and that of "form" as "shape". We can think of the *form* as being the appearance of the object, while the *shape* is extracted from the *form*, by removing the dependence upon global transformations, i.e., scaling, translation, and rotation. Consider, for example, the outlines of the base of a 1×1 and a 2×2 Lego brick. Though their extrinsic forms certainly differ, with one larger than the other, their intrinsic shape remains the same. In this way, mathematically, *shapes* are elements of an infinite dimensional space, as we will discuss further in Section 2.2.3.
Separating the notion of shape from form is especially useful when working on applications with image data. Here, we may not have the size and orientation information of the objects in the images. And moreover, it is the camera shot that affects these attributions. For example, photographs can show the same object but depending on how close the camera was, the objects can appear to have different sizes. In this thesis, where it is relevant, we will separate shape from form, by removing the global similarity transforms. There are well-established tools for this, as will be discussed in the Background chapter.

1.1.1 Shape Data

It has been estimated that by 2025, there will be 175 zettabytes[1] of data in the world (Rydning, 2018). Included in that number will be a plethora of potential *shape data*; in other words, datasets that consist of images of objects or details of objects, from which shape information can be *extracted*. Already, we can see petabytes upon petabytes of data that can be considered as shape data, for instance the petabytes of neuroimaging data produced every year, (Dinov et al., 2014). Thus, from images that show distinct objects, we can consider the shape of said object by its outline. Inside a computer, we may represent a shape by:

Landmarks: Landmark points are defined in the field of morphometrics as points on an object that correspond to definitive points across a population of similar objects. Importantly, landmark points have to preserve their correspondences throughout the dataset. To better illustrate this, we present an example of landmarks distributed on a dataset of leaf images (as seen in (Firmansyah et al., 2016)). In order to grasp a better understanding of the comparisons between the shapes of leaves throughout the dataset, landmarks

[1]Where 1 zettabyte is equivalent to 10^9 terrabytes.

can be placed on anatomical points such as on the apex (*tip*) of the leaf, its base, or its petiole (*stalk*), where in this example, we make the assumption that these features exists on each of the leaves. Landmarks can also be defined in a more *quantitative* way, for example, by finding points on an outline that correspond to the points of highest curvature.

Curves: Instead of selecting representative points, we can describe shapes as continuous functions of some underlying parameter. For a two-dimensional curve, one parameter is sufficient. The curve starts at a point a, and ends at a point b, where $a, b \in \mathbb{R}$, and the curve, c, is a mapping such that $c : I = [a, b] \mapsto \mathbb{R}^2$. Curves can be open or closed. In the latter case, $c(a) = c(b)$. In order to include this condition, we can instead consider a closed curve, c, as a mapping, $c : \mathbb{S}^1 \mapsto \mathbb{R}^2$. We will go into more detail about these mappings in Section 2.2.1. Curves can be parametrised in many different ways. Note that points on a curve will no longer need to be *in-correspondence*, unlike in the landmarks approach.

1.2 Analysing Shape Data

Owing to the myriad of shape data readily available today, it is now more important than ever to establish ways in which to reliably analyse the data to its fullest.

1.2.1 Morphometrics

The hisotry of analysing shapes is often attributed with the field of morphometrics. This is not too surprising, as the word morphometrics comes from "*morphe*" meaning "shape", and "*metron*", meaning "measure". There is a long history of analysing the shape of objects with morphometrics, such as the anthropological studies of facial structures in the late 1800s, as described in (Mitteroecker and Gunz, 2009). These methods often employed *Bookstein Shape Coordinates*[2], (Bookstein et al., 1985), to describe transformations between landmarks; a technique still used today, such as in the palaeontological study on predation analysis in (Leighton, 2011). By the late 20^{th} centuary, the applied mathematician, Fred Bookstein, embarked on using Cartesian coordinates to describe shapes with

[2]Often considered the simplest tool in morphometrics, involving the removal of similarity transforms.

landmarks, (Bookstein, 1997), and opened the doors to the area of Geometric Morphometrics, which is arguably, one of the most popular approaches in the analysis of shapes, particularly within anthropology, zoology, palaeontology, and archaeology. Geometric Morphometrics focuses on the statistical analysis[3] of shapes or forms of objects, as described by the Cartesian coordinates of the object's landmarks. Moreover, the analysis is done whilst preserving the geometric relationships of the landmarks in question; this is an important difference between this approach and traditional methods of morphometrics. For more details regarding Geometric Morphometrics, we refer the reader to (Bookstein, 1997), which provides a useful analysis on morphometric tools and to (Rohlf and Marcus, 1993), and (Adams et al., 2004), which describe the *revolution* in morphometrics.

1.2.2 Shape Analysis & Functional Data Analysis

One place where methods of Geometric Morphometrics may fall short is on their reliance on landmarks. Though landmarks are a convenient tool in the analysis of shapes, subtle changes in them can sometimes lead to inaccuracies in analysis, for example, if there are too few landmarks, or if the positions of the landmarks are not in correspondence across a dataset. These particular problems are partially handled by using a *semilandmark* approach, at the cost of optimisation, as discussed in (Gunz and Mitteroecker, 2013), a study that introduced the concept of *semilandmarks* on curves and surfaces.

Another potential disadvantage of Geometric Morphometrics methods is the occasional incorporation of standard linear methods for further statistical analysis, even when the space of the shapes is not Euclidean. Such approaches risk painting a picture of the shapes of objects, that may not be mathematically as accurate as it could be.

There are many possible issues that could arise form landmark-based methods, from processing time due to the landmark-selection step, to the issues already mentioned here. For more information, we refer readers to (Marsland and Shardlow, 2017), for a concise description of landmark-based analysis of shapes with uncertainties. Furthermore, we note that these possible deficiencies in landmark-based methods are something we will be exploring in this thesis, namely in Section 4.1 of Chapter 4. This brings us on nicely to a field that overcomes such issues, namely, (mathematical) shape analysis.

[3]such as Principal Component Analysis, PCA.

Shape analysis can be loosely thought of as the mathematical study of shapes. It is part of Functional Data Analysis, which deals with functional data, as the name suggestions, i.e. data that can be considered as real-valued functions on fixed intervals, such as growth curves. Henceforth, the statistical tools that allow us to analyse shapes (or *functional data*) developed in this field, can be applied to numerous real-world problems, from facial recognition to medical imaging.

The origin of shape shape analysis is often aligned with the publication of D'Arcy Thompson's notable book "On Growth and Form" (Thompson, 1917). This pioneering book (which we shall go into more detail about in the next chapter) was the first to present the idea of comparing two shapes for evolutionary analysis, by deforming a grid; a method once proposed by the painter and mathematician, Albrecht Dürer in the 1500s, in his work on human proportions. More specifically, by drawing an object onto a grid, one searches for a means of *morphing* the grid so that the original shape of the object is deformed in such a way that it resembles the other shape. This *morphing* from one shape to another can be referred to as a *matching*, and plays a significant role in functional and shape data analysis. By *morphing* a shape in such a way, we can incorporate a sense of elasticity, that can perceivably provide a shape with more opportunities and freedom for it to be morphed into another. This is the motivation for elastic (shape) matching.

1.2.3 Elastic Matching

As the title of this thesis suggests, our interest lies predominantly within the *elastic matching* of shapes. Illustratively, we can think of an elastic matching between two two-dimensional curves by imagining one curve being drawn on an elastic sheet, and that sheet being stretched and compressed until the deformed shape resembles the other curve. In the field of shape analysis (or more precisely in this case, *elastic* shape analysis), these deformations between shapes, caused by an underlying *grid* deformation, are the results of the action of diffeomorphisms[4] on the shape itself. For example, we consider the shapes of two closed curves, $c_1, c_2 : \mathbb{S}^1 \mapsto \mathbb{R}^2$. The objective is to find a diffeomorphism, $\phi : \mathbb{R}^2 \mapsto \mathbb{R}^2$ such that its action can be defined to describe a *matching* between the two curves, where (ϕ, c_1) is *matched* with c_2 and the action $(\phi, \cdot)$ is by composition.

[4]A formal definition of diffeomorphisms will be provided in Chapter 2. For now, we can think of these as *nice*, invertible transformations.

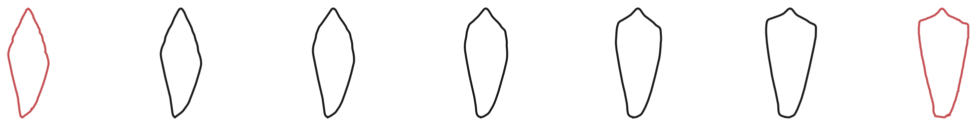


Figure 1.1: Elasting matching between the shell outlines of two cone snails (*Conus*). The matching is illustrated over *time*, via geodesics, as described in the next chapter.

An example of an elastic matching is shown in Figure 1.1, where a closed curve representing an outline of a shell is *matched* with another shell within the *Conidae* family. Elastic matchings are not only helpful in illustrating *how* a shape can be morphed into another, but more importantly, they can be used to define a distance between two shapes, for example, by computing the *energy* required to perform the deformation. Such *distances* represent the quantification of differences between the shapes of two objects, which can be highly useful in statistical analysis of shape data.

1.3 Our Research

There are a vast variety of applications where the shapes of objects are relevant, from outline curves of objects to handwriting recognition to any function that can be graphed as a continuous curve. Just within landmark analysis using morphometrics, more datasets are produced regularly, particularly in the field of biology and ecology, such as in (Gündemir et al., 2022), an analysis on the shapes within sternums of different birds. Despite this vastness, real-world studies that utilize elastic shape matching and other tools from elastic shape analysis, are rare. Therefore to showcase these opportunities, this thesis is comprised of multiple projects that each work on differing types of data, which we will soon see. By working on this anthology of different projects, we can provide a highly useful and detailed overview of the applications of elastic shape analysis on two-dimensional curves.

Broadly, our projects can be split into two categories based on the types of their data, specifically: image-based data and non-image data. Whilst in the image datasets we are making an assumption that the datasets in our objects have an *outline*, in the latter, we are working with a natural definition of a curve, such as a growth curve. In this section, we discuss the motivation behind these projects, their datasets, and our objectives.

1.3.1 Motivation & Objectives

Shape analysis methods have been around for a few decades, from methods used to separate shape from form, such as Procrustes alignment (see Chapter 2), or methods of temporal curve alignment, as in the Dynamic Time Warping method, (Berndt and Clifford, 1994). However, our focus is predominantly on the newer and less-explored field of *elastic* shape analysis, which we believe has yet to see its full potential in practical applications. Collaborations involving elastic shape analysis on varying datasets and alongside researchers outside of the field, are seldom seen, hence the inclination of a greater potential to these methods, and the motivation to build the necessary bridges to explore the possible practical applications. Furthermore, based on both the theoretical and practical developments in elastic shape analysis in recent years, shape methods are now mature enough to be applied and used in real-world projects. Therefore, it is now the ideal time to be exploring such applications, starting, in this thesis, with the applications of elastic shape matching on two-dimensional curves.

Our global ambition is to make elastic matching for the analysis of shape data accessible to a much wider audience. To accomplish this, we undertake projects that analyse shape data, via a data science lens, and in turn, build a framework that can be applied to a broad range of questions and datasets, in a broad range of fields. Henceforth, this research is based on a collection of aims, objectives, and questions, that can be split into different themes. In the following section we mention the various new contributions of this doctoral research by describing the themes that motivated this work, that are the building blocks to this thesis.

Interdisciplinary Collaborations We want to apply elastic shape analysis to real-world projects in collaboration with experts from a wide variety of fields and across several datasets, from natural objects, such as (i) growth curves[5] in collaboration with conservationists and ecologists; (ii) shell datasets, in collaboration with malacologists[6]; (iii) to analysis on ancient vases, in collaboration with archaeologists and historians. By working directly with experts in the relevant fields, the results of our projects can be properly evaluated, have immediate effects and be utilized to make a true difference to the area. In particular, we collaborate with evolutionary biologist, Professor Armand Leroi[7], at Im-

[5]more specifically, as we will see in the penultimate chapter, our growth curves are weights over time.

[6]Malacology is the study of Mollusca e.g., snails and cephalopods, as well as other invertebrates.

[7]`https://www.imperial.ac.uk/people/a.leroi`

perial College London, for our work on shapes of ancient Greek vases, gastropod shells, and Swedish leaves (– Section 4.1). We collaborate with Dr Andrew Digby[8], ecologist and scientific advisor at Kākāpō Recovery[9], on our project Kākāpō health project (– Section 5.1). We collaborate with marine biologist, Professor Jonathan Gardner[10], from Victoria University of Wellington, on our Mollusc classification project (– Section 4.3). We collaborate with the AviaNZ[11] team, at Victoria University of Wellington, on our project relating to within-species birdsong identification (– Section 5.2).

Significant Datasets The datasets we will be working with cover a range of areas and provide a useful account of the applications of elastic shape analysis. Not only are these datasets useful in their variation, but they are of great importance too. For example, our work in the final chapter of this thesis is focused on possible applications in wildlife conservation. And in one particular project, we will be working with a biological dataset from a highly endangered species. Thus, our projects are not only useful in proving an overview of the possibilities of elastic shape analysis applications, but they also present an exciting opportunity to work with important datasets that have not been studied before. Furthermore, as an additional side-note, throughout this thesis we will try our best to provide links to every dataset used or mentioned, as we recognise the importance of sharing data for the progression of science.

Analysis of Distances The primary output from the elastic shape analysis stage of our projects is a set of *distances* between shapes in the dataset. Subsequently, the aim is often to analyse these results by delving into tools from machine learning. This is different to most data-science-related research, which derive features from objects that are then incorporated into machine learning algorithms that operate in $\mathbb{R}^n$, whether that's via a supervised or non-supervised learning approach. Whilst certain deep learning methods such as Convolutional Neural Networks[12] side-step the requirement of the user to pre-select the best features, their dependence on very large training data, and their often lengthy training and application run-time, means that we cannot incorporate them in

[8]https://www.doc.govt.nz/our-work/kakapo-recovery/meet-the-people/kakapo-recovery-team/
[9]https://www.doc.govt.nz/our-work/kakapo-recovery/
[10]https://people.wgtn.ac.nz/Jonathan.Gardner
[11]https://www.avianz.net/
[12]which we will briefly describe in Chapter 3.

our specific projects. Furthermore, many machine learning methods[13], such as Support Vector Machines, (Cortes and Vapnik, 1995) or K-means clustering, (Lloyd, 1982), in general, work directly with feature vectors that describe objects in a dataset, and then transform the space where a distance metric (typically, linear) is subsequently defined. However, since we don't have *features*, we want to use a distance matrix instead, and thus, we adapt the relevant machine learning algorithms specifically for our projects, in order to incorporate our distances or our distance metric.

Classification of Shapes Classification analyses are becoming increasingly important, especially as we globally aim to automate many processes, from automatic recognition of objects in images (such as animals in camera traps, as seen in (Schneider et al., 2018)), to machine learning for classifying breast cancer in (Amrane et al., 2018). Classification studies, particularly in medicine, biology, and ecology (to name a few) often use the shape of an object as one of the attributes. This motivates the question, *can a classification algorithm be trained to classify objects, where the object's shape is its sole attribute?*. In several projects across this thesis, we will explore this question and analyse the performance of classification based on distance matrices that solely contain the distances between the shapes of objects within the datasets. Moreover in our final project, we examine the effect of incorporate additional non-shape-related information alongside the distance matrices that contained the elastic matching results.

Comparisons with Popular Methods We want methods of elastic shape analysis to become popular tools for analysing shape data <u>outside</u> of the shape analysis community. For our methods to be tried by others, we must see how they compare with other popular methods that analyse shape data. Whilst in many areas, geometric morphometrics methods are the most widely-used, in other areas, objects and their shapes and forms are often analysed by eye. Thus, in Section 4.1, we compare multiple elastic shape analysis methods with common methods used in other fields, on varying datasets.

Extraction of Shape Data We will exclusively be working with two-dimensional open and closed curves, as it is the *shape* of the objects that we are solely interested in. Thus, when it comes to image data, it is the outline curves within the image that we are in-

[13]We will go into more detail about certain machine learning methods later on in this thesis. But for a general overview of machine learning alrogirthms, see (Marsland, 2015) or (Getoor and Taskar, 2019).

terested in, not the image as a whole[14]. Since we are now describing three-dimensional objects by their two-dimensional outlines, one project in our thesis (Section 4.2) examines whether this may harm performances when applied to classification questions. Furthermore, as our focus is on the shape of the objects, extracting the outline curves is a highly important step that is, more often than not, glossed over in literature regarding the analysis of shapes. Our research involves finding reliable methods to obtain smooth outline curves from image data, regardless of the quality of the image. In chapter 3, we introduce a novel algorithm that we created and a framework that can be used to automatically extract smooth contours from images reliably.

We hope that the contents of this thesis can be regarded as a concise framework to be employed when analysing shape data via elastic matching. Our research not only explores the mathematics of elastic matching, but also investigates the data extraction and the machine learning analysis, which are just as important when it comes to analysing shape data. By combining all three segments, our work, including our accompanying code, can be viewed as a guidebook to analyse shape data, no matter what experience one has in shape analysis, computer vision, or machine learning. That is one of the fundamental objectives of this PhD and this thesis.

1.3.2 Datasets

As previously mentioned, our research is built upon a combination of multiple projects that are each based on different datasets. These include the following:

- **Greek Vases:** Ancient Greek vase images, of varying quality, from a well-known pottery database, the Beazley Archive[15]. In addition to vase images found here, we also work with first-hand data that our collaborators have obtained. These include photographs of modern-day Greek vases taken in Lesbos, Greece; outlines of Iron-Age Greek vases measured by hand; and 3D laser scans of *lekythoi* vases.

- **Gastropod Shells:** High quality images of gastropod shells of various species, obtained from Gastropods.com[16], taken from their *front* profiles.

[14]Though it can certainly be useful sometimes to analyse images as a whole (for example, in applications of medical imaging), given the additional texture information that images comes with, however, we will not be including them in this thesis

[15]`https://www.carc.ox.ac.uk/carc/pottery`

[16]`https://conchology.be/?t=261`

- **Leaves:** Images of Swedish leaves from the Swedish Leaf Dataset[17] - a well-used dataset both within the field of elastic shape analysis, such as in (Laga et al., 2014), a paper on landmark free methods to analyse leaf shapes, and outside, such as the leaf classification study in (Mouine et al., 2013).

- **Kākāpō Growth Curves:** Database obtained from our collaborators at Kākāpō Recovery[18] with data regarding the health of all kākāpō, alive and deceased, that have been monitored in the last few decades. Information includes location, sex, and multiple weights, where the latter are used to create growth curves.

- **Kiwi Calls:** Audio recordings of kiwi calls from sound recorders left in the wild, that are managed and monitored by our collaborators at AviaNZ[19].

- **Mussels:** Images of mussels from the North Atlantic, the Mediterranean, and mainland New Zealand as well as its offshore islands. The images were collated by our colleagues at Victoria University of Wellington.

1.4 Thesis Outline

The remaining chapters of this thesis begin with an introduction to elastic shape analysis, followed by an overview of image processing methods. From here, we dive into the numerous applications that our research is built upon. These projects have been grouped into two chapters, based on the *type* of the primary datasets involved.

2 – **Background Chapter**
This chapter begins with an overview of the field of elastic shape analysis, recent literature, and its history. Subsequently, we focus on *how* a curve can be transformed into another. To achieve this, we briefly study the roles of diffeomorphisms, and geodesics, which describe shortest paths between points. Finally, we delve into implementations that allow us to match / align both open and closed curves in $\mathbb{R}^2$.

3 – **Image Data Processing**
In this chapter, we look at some popular algorithms in Computer Vision, with the aim of extracting curves from image data. This chapter is broadly split it into

[17]`https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/`
[18]`https://www.doc.govt.nz/our-work/kakapo-recovery/`
[19]`https://www.avianz.net/`

sections on image binarization and (outline) contour extraction. The purpose of the former is to create black and white versions of images, in order to improve the performance of the contour extraction algorithms. We study some common algorithms used in Computer Vision and used in our research, and will also introduce an image binarization algorithm that we created specifically for our projects.

4 – **Applications of Elastic Shape Analysis to Closed Curves Extracted from Images**
The majority of this chapter is focused on one large collaborative project regarding the classification of organisms and artefacts by their shapes. Here, we compare various methods of Elastic Shape Analysis and Geometric Morphometrics to analyse the shapes found in image data consisting of Ancient Greek vases, gastropod shells, and Swedish leaves. Additional projects in this chapter include a Mussel identification project, and an analysis on the accuracy of shape outlines found in vase images, in comparison to hand-measured outlines and 3D laser scans.

5 – **Applications of Elastic Shape Analysis on Open Curves**
Here, we take a look at two collaborative projects aimed at utilizing elastic shape analysis in real-world conservation; in particular, the conservation of some of New Zealand's most remarkable birds. In one project, we create a novel tool which incorporates elastic shape matching to predict the health of kākāpō chicks, in collaboration with conservationists and ecologists at Kākāpō Recovery[20]. Meanwhile, in the second project, we explore the applications of elastic shape analysis to raw sound data, something that to the best of our knowledge, has never been done before. This project aims to classify individual kiwi, based on the *shapes* of their calls, through the use of spectrograms, but could be used more widely.

6 – **Summary & Conclusion**
We summarize the results of our projects and discuss their future. And finally, we'll explore further possibilities for real-world applications of elastic shape matching.

[20]https://www.doc.govt.nz/our-work/kakapo-recovery/

Chapter 2

Background

2.1 Introduction

From a surface describing the beak of a spoonbill to a two-dimensional population growth curve of a country, or an MRI scan of a heart to the outlines of hand-written characters - there is a lot we can learn from the shapes of objects. This field of research is known as shape analysis: the mathematical study of shapes.

We begin this chapter with a brief introduction to shape analysis, where we delve into the history of shape analysis, visit some *long-established* methods, and review recent literature. We then outline the contents of the remainder of this chapter, which is dedicated to an overview of elastic shape analysis, and in particular, elastic matchings between curves.

2.1.1 History & Literature Review

Over a century ago, in 1917, the mathematician and biologist D'Arcy Wentworth Thompson published a pioneering book titled "On Growth and Form" (Thompson, 1917). Amongst many concepts, the book included a study of the shapes of animals and plants. It was ground-making in its introduction to mathematical biology and paved the way for further topics such as computational anatomy and consequently, shape analysis. Notably, the book alluded to allometry[21], and discussed the use of mathematical transformations in the evolution of the shape of species. An example of such transformation can be seen

[21]Allometry is broadly the study of the relationships between the size, shape and anatomy of living organisms, and their impacts on evolution, ecology and so on. Moreover, it was first alluded to by Otto Snell in (Snell, 1892).

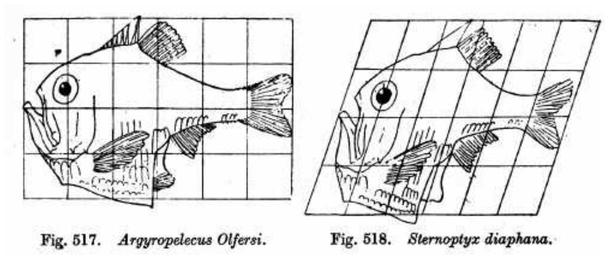


Fig. 517. Argyropelecus Olfersi.
Fig. 518. Sternoptyx diaphana.

Figure 2.1: An illustration of a transformation of a ray-finned fish using a shear mapping, taken from On Growth and Form, (Thompson, 1917).

in Figure 2.1. Thompson showed that by drawing an *Argyropelecus olfersi* (a genus of ray-finned fish) on a Cartesian grid, and applying a shear map to transform the points on the drawing in a certain direction, the *Argyropelecus olfersi* can be transformed and hence begin to resemble another type of ray-finned fish, the *Sternoptyx*.

Whilst Thompson's work focused more on the *forms* of objects, our interest is on *shape*. Though we will go on to provide a more mathematical definition of *shape* with each of our applications, we can broadly consider the shape of an object as its form, modulo global transformations such as scaling, rotation, and translation. By separating shape from form, we can analyse the shapes of objects irrespective of such differences, in order to grasp a better understanding of the shapes. This was particularly highlighted by David Kendall, who pioneered the study of shapes within their shape spaces, and considered the global transformations that affected form, but not shape, as nuisance parameters (Kendall, 1984).

The twentieth century brought great advances, not only in shape analysis, but also in functional data analysis in general. From Kendall's work on shape spaces, (Kendall, 1984), Bookstein's analysis on landmarks with Bookstein coordinates (as previously mentioned in Section 1.2.1) (Bookstein and Sampson, 1990), or methods of Procrustes alignent ((Sibson, 1978), (Gower, 1971)), to statistical analysis using maximum likelihood estimations, as seen in (Mardia and Dryden, 1989).

As we'll soon discover throughout this thesis, our main interests are on two (connected) aspects of shape and functional data analysis: namely, aligning shapes, or more widely, functional data, (as shown in Figure 2.2) and quantifying the differences between two shapes by computing a *distance*. In the last few decades, we have seen an array of methods to perform such things, from the Dynamic Time Warping method in (Berndt and Clifford, 1994), used to warp time series data such as population growth curves, to Geodesic Distance Analysis in (Joshi et al., 2011), or a *precise* dynamic programming method to compute distances between open curves in (Lahiri et al., 2015).

Recall that in this thesis, our sole focus will be on two dimensional curves. One way we can analyse our curves, is with the use of elastic-like transformations based on diffeomorphisms. The incorporation of diffeomorphisms leads to the potential of performing more *accurate* transformations and distance computations between shapes, that may have been less pertinent with previous landmark-focused or grid-based methods. In the last decade or so, we have seen developments of such diffeomorphic methods, which form the basis of elastic shape analysis. Concurrently, we have also seen an increased focus on the applications of elastic shape analysis methods, such as biomedical applications as seen in (Bharath et al., 2018), an image analysis of brain tumours, (Cho et al., 2019) a broad tutorial of elastic shape analysis on biological structures, from leaf outlines to DT-MRI fibres, and also (Laborde et al., 2011), a study aligning the structures of RNA molecules using elastic shape analysis.

Literature on the applications of elastic shape analysis has varied in recent years. For example, in (Eslitzbichler, 2015), we saw the applications of shape analysis on motion capturing. Here, the author utlized Square Root Velocity (SRV) representations[22] of animations curves, to approximate cyclic animations. Furthermore, they incorporated geodesic distances to classify animations[23]. Other examples of applications of shape analysis on animation includes the research seen in (Bauer et al., 2015), which was also based on elastic matching of skeletal animations, within an SRV framework.

Amidst the literature, many works discussing such applications have focused on well-known and *well-used* datasets, such as the Flavia Leaf Dataset[24], the Swedish Leaf Dataset[25], the Kimia shape database[26], Unipen handwriting data[27], and the Berkeley

[22]We will discuss the square root velocity functions later on, in Section 2.4.2.

[23]For example, to distinguish actions such as walking and running, and so on.

[24]`http://flavia.sourceforge.net/`

[25]`https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/`

[26]`https://github.com/mmssouza/kimia99`

[27]`http://www.unipen.org/home.html`

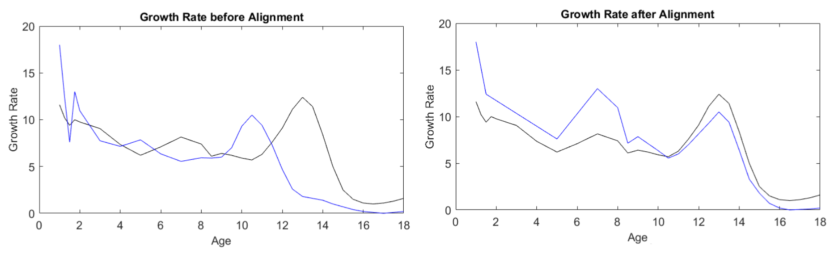


Figure 2.2: An alignment between two growth rate curves. On the left we see the two curves before any alignment has taken place, whilst on the right we see a plot after alignment. Here, the blue curve has been aligned to the black curve using a Dynamic Programming Algorithm, which we will encounter later on in section 2.5.2.

Growth dataset[28], to name a few. For example, handwriting applications were explored in (Kurtek and Srivastava, 2014), in order to successfully[29] segment letters and words using elastic matchings.

Despite the volume of existing work involving elastic shape analysis, the methods have not seen much uptake: analyses are still primarily in the papers where a method is developed, and more often than not, with *clean*[30] curves, originating from well-known datasets. Hence a *framework* has yet to be introduced, detailing the process of applying elastic shape analysis, from start to finish. Furthermore, there is an absence in literature, when it comes to comparing different methods of elastic shape analysis in practical applications, as well as comparisons with Geometric Morphometrics. This is the motivation behind this thesis.

2.1.2 Procrustes Alignment

Outside of Shape Analysis, scientists often associate alignments between shapes with the well-known Procrustes[31] alignment method. Procrustes alignment, also known as Procrustes superimposition, is the most widely-known shape analysis method. It has its roots back in the early 20th century, such as in (Mosier, 1939). We briefly outline the method, and refer readers to (Goodall, 1991) for more details.

[28]https://doi.org/10.7910/DVN/OWCPHT

[29]In particular, one small experiment on segmenting characters from words, had a 96% success rate.

[30]by which we mean data that has already been processed, and is unlikely to have much noise.

[31]Procrustes Alignment is rather gruesomely named after a character from Greek mythology, *Procrustes*, an inn-keeper who supposedly stretched, compressed, or simply cut off limbs from his victims, in order to fit them into the beds of his inn.

Procrustes alignment can be used to optimally rotate, translate, scale, and reflect shapes, to match some template. As an example, take two curves $c_1, c_2 : I \mapsto \mathbb{R}^2$, for some interval I on a real space, with c_1 chosen as the arbitrary template. We consider our curves to be discretized at n points thus our curves are just a set of coordinates that can be written $c_i = \{(x_1^i, y_1^i), ..., (x_n^i, y_n^i)\}$, for $i \in \{1, 2\}$.

The first steps in Procrustes alignment is the removal of translation and scaling variability. The simplest way to do this is to translate the curves so that they lie at the origin. Let's take either one of our curves. For a discretized curve $c = \{(x_1, y_1), ..., (x_n, y_n)\}$, we can find the mean curve $\bar{c}$. This can then be used to find the translated curve c_{tr}, and hence the scaled curve c_{sc}, which is scaled to be of unit-length:

$$\bar{c} = (\bar{c}_x, \bar{c}_y) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i \right)$$

$$c_{tr} = \{(x_1 - \bar{c}_x, y_1 - \bar{c}_y), ..., (x_n - \bar{c}_x, y_n - \bar{c}_y)\} \tag{2.1}$$

$$c_{sc} = \frac{c_{tr}}{\sqrt{\sum_i^n \left((x_i - \bar{c}_x)^2 + (y_i - \bar{c}_y)^2 \right)}} \tag{2.2}$$

When it comes to rotation, we fix the template curve and rotate the other around the origin, so that the sum of squared distances between it and the template are minimized. Note that here, we use the *version* of the curves that have had the translation and scaling variability removed. The rotation matrix can be defined as follows:

$$R_{rot}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{2.3}$$

If we consider our transformed curves in matrix form, say with M_1, M_2, the we compute the rotation of one, M_2, such that:

$$R_{rot} M_2 = VU^T \text{ , where } M_2^T M_1 = UDV^T \tag{2.4}$$

using singular value decomposition to find the diagonal matrix D, and the matrices U, V. And the *optimal* angle, θ^*, is the result of the minimization problem:

$$\theta^* = \arg\min_{\theta} \sum \left(R(\theta) M_2 - M_1 \right)^2 \tag{2.5}$$

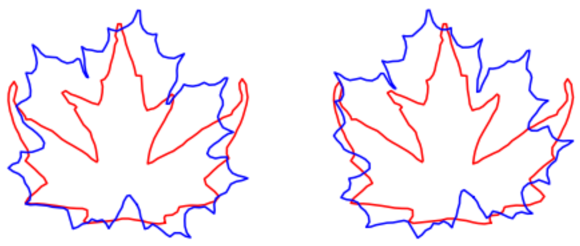


Figure 2.3: Procrustes alignment of leaves. Here, we see an outline of a *modern-day* leaf (blue), being aligned to an outline of an ancestral Miocene leaf (red). The left plot shows the original two leaf outlines, whilst the right plot shows the leaf outlines after the blue curve has been aligned with the red using Procrustes alignment.

Note that a rotation of curves is simply the composition of two *reflections*. Thus, just as we defined our rotation matrix with an angle θ in (2.3), we can define a reflection matrix, R_{ref}, with angle 2θ:

$$R_{ref}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{2.6}$$

Henceforth, to find an optimal reflection, we once again perform the minimization as seen in Equation 2.5, but with R_{ref}, in order to find the optimal angle.

An example of a Procrustes alignment can be seen in Figure 2.3, where an outline of a leaf (blue) has been aligned with an outline of ancestral leaf (red) using Procrustes alignment.

The computational simplicity of Procrustes alignment make it a worthy choice for a basic alignment. Henceforth, it is often used as a pre-processing step for further analysis of sets of shapes, of the same coordinate system, to extract *shape* from *form*. Furthermore, though we do not focus too much on Procrustes alignment in this thesis, since the methods we employ are invariant to such similarity transforms, as we'll see in subsequent chapters, we do sometimes incorporate it in certain projects, as a pre-processing step, when required (as is the case Section 4.1).

2.1.3 Background Chapter Outline

Throughout this chapter, we provide an overview of elastic shape analysis on two dimensional curves. We outline the definitions of shapes and shape spaces, and explore how a shape can be transformed into another, whilst a *distance* is computed between them. We then take a look at various implementations of methods of elastic shape analysis that we utilize in our projects in the subsequent chapters of this thesis; with a focus on one framework in particular: the Square Root Velocity framework, (Srivastava et al., 2011b). The subsequent sections of this chapter are as follows:

2. **Shapes & Invariances** – We begin by defining a shape, which involves discussing invariances to shape-preserving transformations, including reparametrization.

3. **Transforming Shapes** – In this section, we explore how a shape can be transformed into another, and how this can be used to quantify differences between them. We focus on LDDMM (i.e., Large Deformation Diffeomorphic Metric Mapping) - a widely-used framework in elastic shape analysis, and outline other diffeomorphic methods that we incorporate in our projects.

4. **Square Root Velocity Framework** – Here, we focus on arguably the most important framework in this thesis, the SRV (or SRVF) Framework. The framework is based on the usage of a certain representation of curves, known as square root velocity functions, which we will fully explore in this section.

5. **Implementations** – Finally, we take a look at implementations of methods discussed in previous sections. We also, evaluate a small project that was designed to compare various methods of alignment between open curves in $\mathbb{R}^2$.

2.2 Shapes & Invariances

In this section, we define what *shape* means to us, in this thesis, and subsequently, we explore what the space of these shapes looks like.

2.2.1 Parametrising Curves

As we'll discover in the succeeding chapters, the majority of our projects are concerned with the two-dimensional outlines of objects found in images. Regardless, our interest is always on two-dimensional curves, either open or closed.

Curves can be parametrized in various ways. In the simplest case, we can consider discrete points along an object's boundary. Here, we can describe the boundary curve, c, with N

points, for example:

$$c_{\text{points}} := \{(x_i, y_i) \in \mathbb{R} \times \mathbb{R} = \mathbb{R}^2 \mid i = 1, \cdots, N\}. \tag{2.7}$$

Alternatively, we can focus on functional data (such as growth curves). Here, we are concerned with functions on fixed intervals. For example, let's consider discrete points representing a growth curve over a fixed-time domain, $I = [a, b]$. We start with a description of the curve as $c \subseteq ([a, b] \times \mathbb{R})$, where $a, b \in \mathbb{R}$. The next step is to find a way to estimate the function, $c(t)$, from these discrete points. A simple way would be with a piecewise linear curve[32] representation. Or, for a potentially *smoother* estimation, we can employ a Least-Squares approach, where we can define a function $c(t)$ by a linear combination of points, using some basis function that belong to a Hilbert space with a given inner product structure, as described in Chapter 4 of (Srivastava and Klassen, 2016).

For the remainder of this chapter, we'll consider open and closed curves as continuous mappings $c : I \mapsto \mathbb{R}^2$, on some real interval $I = [a, b]$. We note that when considering closed curves, we require the additional condition that the curve *starts* and *ends* at the same point, i.e. $c(a) = c(b)$. We can describe a closed curve c, as a mapping from the unit circle $S^1 := \{x \in \mathbb{R}^2 \mid ||x|| = 1\}$, such that $c : \mathbb{S}^1 \mapsto \mathbb{R}^2$. This leads to a natural parameterisation by arc-length: a curve runs from $\theta = 0$ to $\theta = 2\pi$. In this thesis, we will consider curves that do not have any self-intersections. This additional self-intersection property leads us to a more specific definition of the curves that we will focus on. First, we begin with defining a function often seen in the field of topology: homeomorphisms.

Definition 2.1 (Continuous function) *A function $f : X \mapsto Y$, where X and Y are topological spaces, is continuous at a point $x \in X$, if and only if $\forall$ neighbourhoods $V \subset Y$, of $f(x)$, $\exists$ a neighbourhood of x, U, where $f(U) \subseteq V$.*

Definition 2.2 (Homeomorphism) *$f : X \mapsto Y$, where X, Y are topological spaces, is a homeomorphism if it is continuous, bijective, and has a continuous inverse.*

Importantly, we can now define what an embedding is:

Definition 2.3 (Embedding) *An embedding is a homeomorphism onto its own image.*

Therefore in this thesis, we describe a closed, non-self-intersecting curve c as embedding, $c \in \text{Emb}(\mathbb{S}^1, \mathbb{R}^2)$, where the self-intersection condition is implicit from the bijective property of homeomorphisms.

[32]This can be illustrated by drawing *straight-lines* between (x_{i-1}, y_{i-1}) and (x_i, y_i) $\forall i = 2, \cdots, N$.

2.2.2 Shape-Preserving Transformations

We are interested in the *shapes* of objects. This leads us to two questions:

1. Are there curves that have the same shape, but look different?
2. Are there curves that are parametrised differently, but look the same?

At the start of this chapter, the first question was somewhat referenced when we discussed separating *shape* from *form*. For example, let's consider two curves, where one represents an outline of a violin, c_1, and the other is a scalar multiple of it, $c_2 = kc_1$ for some non-zero $k \in \mathbb{R}^+$. Though these two curves may look different, as one is larger than the other, they both have the same violin shape. Scaling is an example of a shape-preserving transformation. Other recognizable shape-preserving transformations include translation (e.g. curves $c_1(t) : \mathbb{S}^1 \mapsto \mathbb{R}^2$ and $c_2(t) : \mathbb{S}^1 \mapsto \mathbb{R}^2$, where $c_2(t) = c_1(t-a) + b$ for some $a, b, \in \mathbb{R}$ would have the same shape), and rotation (e.g. $c_1, c_2 = R_{rot}(\theta)c_1$ for some angle $|\theta| < 2\pi$, and rotation matrix R_{rot}, as seen in Equation (2.3)). Furthermore, we note that the set of all such rotations can be described with the group[33], $SO(2)$. Finally, note that these transformations can be removed by Procrustes alignment.

Diffeomorphisms

The second question refers to a slightly less *obvious* shape-preserving transformation: reparametrization. For example, take a curve $c_1(t)$. We can reparametrize $c_1(t)$ via a smooth function γ, such that the reparametrized curve $c_2(t) = c_1(\gamma(t))$ has the same shape as $c_1(t)$, where γ is defined as a smooth, invertible, orientation-preserving mapping.

We can label the space of all such reparametrization functions as Γ. More specifically, when considering closed curves, these smooth, invertible, orientation-preserving mappings can be described by diffeomorphisms:

Definition 2.4 (Manifold) *A topological space M is a manifold if it is locally Euclidean (i.e., locally homeomorphic to a Euclidean space).*

Definition 2.5 (C^∞ (Smooth)) *A function that is differentiable for all orders of differentiation is a C^∞ function or a smooth function.*

[33]Recall that a rotation is a member of the special orthogonal group $SO(n) = \{A \in GL(n, \mathbb{R}^2) \big| A^T A = 1_n,\ \det(A) = 1\}$, where $GL(n, \mathbb{R})$ is the general linear group over $\mathbb{R}$.

Definition 2.6 (Diffeomorphism) *An invertible mapping between smooth manifolds, where both itself and its inverse are smooth and differentiable.*

Any diffeomorphism $\pi : \text{Diff}(\mathbb{S}^1) \mapsto \mathbb{S}^1$ modifies the parameterisation of the curve, but not its appearance. Thus the space of reparametrization functions Γ can be described as the space of all such diffeomorphisms π: $\Gamma = \text{Diff}(\mathbb{S}^1)$. In Section 2.4.3, we elaborate on this equality by showing that the set of reparametrizations is a group and explicitly defining what a right group action on it this. For now, for continuity, we will simply say that the group action here is by composition, $\circ$:

Definition 2.7 (Right group action) *Let G be a group, with identity element, g_{id}. A function of G on a set X is defined as a right group action, $\cdot : X \times G \mapsto G$, if the following conditions hold $\forall x \in X$:*

1. $x \cdot g_{id} = x$,
2. $(x \cdot g) \cdot h = x \cdot (gh)$, $\forall\, g, h \in G$.

Definition 2.8 (Function composition – $\circ$) *For two functions f_1, f_2, we define a composition as the operation $\circ$, such that $(f_1 \circ f_2)(x) = f_1(f_2(x))\ \forall\ x \in X$, where X is the domain of f_1.*

For a curve $c : \mathbb{S}^1 \mapsto \mathbb{R}^2$, the right group action: $c \cdot \pi = c \circ \pi$, does not change the image, and hence shape of c. Therefore, we can say that the set of diffeomorphisms, $\text{Diff}(\mathbb{S}^1)$ is precisely the set of reparametrizations, Γ.

2.2.3 Pre-Shape and Shape Space

Recall that the field of shape analysis is the study of shapes, which we broadly described as the forms of objects, modulo certain transformations. In this thesis, these transformations, which we dub as *shape-preserving transformations*, comprise of scaling, translation, rotation, and reparametrization. Therefore, we must define the shapes of our curves in $\mathbb{R}^2$, such that they are invariant to the so-called shape-preserving transformations.

We must first define the space of our open and closed curves. As we are interested in the *shape*, and do not care for size and location, we can easily translate the curves so that they are centred at the origin, and scale them to be unit-length.

$$S_{\text{open}} = \left\{ c \mid c : I = [a, b] \mapsto \mathbb{R}^2,\ \int_I ||\dot{c}(t)||\ \text{dt} = 1 \right\} \tag{2.8}$$

$$S_{\text{closed}} = \left\{ c \mid c \in \text{Emb}(\mathbb{S}^1, \mathbb{R}^2), \int_{\mathbb{S}^1} ||\dot{c}(t)|| \text{ dt} = 1 \right\} \tag{2.9}$$

Let S be the space of unit-length curves of either S_{open} or S_{closed}. Though we have discussed translation and scaling (refer to the last condition of the equations above), we note that we have yet to remove the rotation and reparametrization invariability. In reference to the works of Kendall[34], such as in (Kendall et al., 2009), S is thus defined as the *pre-shape space*. To describe the *shape-space*, $\mathcal{S}$, we must quotient out the effects of the remaining shape-preserving transformations, and to do this, we must first define what an equivalence class is:

Definition 2.9 (Equivalence relation) *A binary relation, $\sim$, on a non-empty set, X, is an equivalence relation if the following conditions, $\forall x \in X$, are satisfied:*

1. $x \sim x$,

2. $x \sim y \implies y \sim x, \forall\, y \in X$,

3. $x \sim y$ and $y \sim z \implies x \sim z, \forall\, y, z \in X$.

Definition 2.10 (Equivalence class) *Let $\sim$ be an equivalence relation on a set X. An equivalence class, $[\,\cdot\,]$, of $x \in X$ is defined as:*

$$[x] = \{y \in X \mid x \sim y\} \tag{2.10}$$

A quotient space can be thought of a set of all equivalence classes and thus, we describe our shape-space, $\mathcal{S}$, as

$$\mathcal{S} = {}^{S}\!/_{\Gamma \times SO(2)}, \tag{2.11}$$

where Γ is the set of reparametrization functions ($\Gamma = \text{Diff}(\mathbb{S}^1)$ when $S = S_{\text{closed}}$). And for a curve $c \in S$, we can define its shape by the equivalence class $[c] \in \mathcal{S}$. Henceforth, *shapes* are simply elements of an infinite-dimensional manifold.

[34]David George Kendall was a mathematician and statistician, known for his pioneering work in statistical shape analysis, amongst other notable work.

2.3 Transforming Shapes

A major theme across this thesis, and, across the field of shape and functional data analysis, is the *matching* of shapes. In other words, the *transformation* of an object's shape into another. This leads us to a notable model in the field of Shape Analysis, namely, the Large Deformation Diffeomorphic Metric Mapping (LDDMM) framework.

2.3.1 LDDMM Framework

The motivation for the matching of shapes is rooted in the myriad of applications it can be utilized within, from pattern recognition to medical imaging and computational anatomy. Indeed, it was with the field of computation anatomy, where the methods of LDDMM truly flourished, as described in (Grenander and Miller, 1998). Here, we provide a general overview of the LDDMM framework on curves, but for more details, we refer the reader to (Beg et al., 2005), where semi-Langrangian methods are incorporated to solve solutions to the LDDMM problem, and (Marsland and Sommer, 2020), which provides a concise overview of topics within geometric shape analysis and the LDDMM framework, with a focus on the diffeomorphism group and its geometries[35].

Action of a Diffeomorphism

The history of LDDMM as we know it today dates back to the work of Beg et al. (2005), which in turn, was based on (Dupuis et al., 1998) and (Trouvé, 1995), as well as the model developed by Christensen et al. (1996). Traditionally, the focus was on transforming a template image, I_0, in order for it to *match* the target image, I_1. Here, a transformation function can be described with $\phi : \mathbb{R}^2 \mapsto \mathbb{R}^2$ (when working with 2D data). The images I_0, I_1 can be defined as $I_0, I_1 : \mathbb{R}^2 \mapsto \mathbb{R}^n$, where n depends on the chosen colour model (for example, $n = 1$ for grey-scaled images, but $n = 3$ when employing the RGB colour model). The action of the transformation on the template can be described by the *pull-back*[36], $\phi \cdot I_0 = I_0 \circ \phi^{-1}$, as explained in (Beg et al., 2005). This is important as it allows to compare the images, without information being missing from the transformed image.

We focus on curves. We define a template curve, c_0, and a target curve, c_1, such that

[35] In this paper, there also explanations of diffeomorphic shape transformations on different examples of shapes, from landmarks to images, and more.

[36] To fit the scope of this thesis, we will simply think of a *pullback* as a precomposition with a function.

$c_0, c_1 \in \text{Emb}(\mathbb{S}^1, \mathbb{R}^2)$. In order to transform c_0, we, once again, define a transformation function $\phi : \mathbb{R}^2 \mapsto \mathbb{R}^2$. Though this transformation mapping can take various forms, to require invertibility (i.e., the existence of an inverse) and smoothness, we restrict ϕ to be a diffeomorphism, as defined in Definition 2.6. We can define a left group action, "$\cdot$", on a curve, by composition, such that $\phi \cdot c_0 = \phi \circ c_0$. Analogous to the idea of stretching and shrinking an elastic sheet with an overlaid curve, in order to *match* the curve with another, the action of ϕ on c_0 is used to *warp* the curve alongside the deformation of the original grid on $\mathbb{R}^2$. Importantly, $\text{Diff}(\mathbb{R}^2)$ is a group, hence due to closure, we can compose diffeomorphisms and obtain new diffeomorphisms.

Theorem 2.11 *Diff*$(\mathbb{R}^2)$ *is a group with group operation given by composition.*

Proof: $\forall\, \omega_1, \omega_2, \omega_3 \in \text{Diff}(\mathbb{R}^2)$:

- **Associativity** - follows from associativity of composition:

$$(\omega_1.\omega_2).\omega_3 = (\omega_1 \circ \omega_2) \circ \omega_3 = \omega_1 \circ \omega_2 \circ \omega_3 = \omega_1 \circ (\omega_2 \circ \omega_3) = \omega_1.(\omega_2.\omega_3)$$

- **Identity** - the identity mapping $id_\omega \in \text{Diff}(\mathbb{R}^2)$ is a diffeomorphism.

- **Inverse** $\forall\, \omega \in \text{Diff}(\Omega)$, $\exists\, \omega^{-1}$ by nature[37], s.t. $\omega.\omega^{-1} = \omega \circ \omega^{-1} = id_\omega$. □

The diffeomorphism group is also analogous to an infinite-dimensional Lie group, (Kriegl and Michor, 1997).

Definition 2.12 (Lie Group) *A group G is defined as a Lie group if it is also a differentiable manifold*[38].

Since the group $\text{Diff}(\mathbb{R}^2)$ is analogous to a Lie Group and Lie groups are smooth manifolds, this leads us to say that it can be endowed with a Riemannian metric[39].

Definition 2.13 (Riemannian Metric) *On a manifold M, a Riemannian metric G is a smooth inner product $G_c : T_cM \times T_cM \mapsto \mathbb{R}$, where T_cM is the tangent space of M at c, $\forall c \in M$. G_c satisfies the following:*

[37] Since the definition of a diffeomorphism requires it to be an <u>invertible</u> mapping.

[38] For the purpose of this section, it is enough to think of a differentiable manifold as a manifold that is additionally globally equipped with a differential structure, which allows the use of calculus on it.

[39] Though we will not go into more details about the mathematics behind this link, as it is outside the scope of this thesis, we recommend (Younes, 1998), a concise paper on elastic distances between shapes, for more information.

1. *$G_c(h,k) = G_c(k,h)\ \forall h,k \in T_cM$*
2. *$G_c(h,h) = 0$ iff $h = 0$*
3. *$G_c(h,h) \geq 0\ \forall h \in T_cM$*

Definition 2.14 (Riemannian manifold) *A smooth manifold M is a Riemannian manifold if it is equipped with a Riemannian metric.*

The Riemannian metric can be used to define a distance, d^*, between two points on a manifold, $M := \text{Diff}(\mathbb{R}^2)$, and hence, a distance between the shapes. For this, the aim if to find the deformation function $\phi \in \text{Diff}(\mathbb{R}^2)$ such that $\phi \cdot c_0 = c_1$. Henceforth, we can sketch an example of a distance between the shapes c_0, c_1, as seen in (Trouvé, 1995):

$$d_S(c_0, c_1) = \inf d^*(\phi_{\text{id}}, \phi), \tag{2.12}$$

where d_S is some distance on the pre-shape space, ϕ_{id} is the identity function in $\text{Diff}(\mathbb{R}^2)$.

The distance between the shapes of the template and target is related to the amount of *energy* required for the deformation. Furthermore, in order for d_S to be symmetric, such that $d_S(c_0, c_1) = d_S(c_1, c_0)$, we restrict it to be right-invariant:

Definition 2.15 (Right-Invariance on *Diff*$(\mathbb{R}^2)$) *A metric $\langle \cdot, \cdot \rangle$ on Diff$(\mathbb{R}^2)$ is said to be right-invariant if $\forall a, b, \phi \in$ Diff$(\mathbb{R}^2)$,*

$$\langle a \circ \phi, b \circ \phi \rangle = \langle a, b \rangle.$$

By requiring our metric to be right-invariant, the distance between points on c_0, c_1 will remain the same, when they are both *deformed* by the same diffeomorphism, $\phi \in \text{Diff}(\mathbb{R}^2)$.

In order to *construct* a right-invariant Riemannian metric, we refer the reader to the previously mentioned paper, (Marsland and Sommer, 2020), which provides a thorough outline of the diffeomorphism group and its geometries. Broadly, the method involves considering an action on the right with $\mathbf{R}_\phi$, for $\phi \in \text{Diff}(\mathbb{R}^2)$, as:

$$\mathbf{R}_\phi : \text{Diff}(\mathbb{R}^2) \mapsto \text{Diff}(\mathbb{R}^2),\ \mathbf{R}_\phi \gamma = \gamma \circ \phi.$$

By the definition of a Riemannian metric, we can work on the tangent space to $\text{Diff}(\mathbb{R}^2)$ at ϕ, which we denote by $T_\phi\text{Diff}(\mathbb{R}^2)$. Henceforth, for $a, b \in T_\phi\text{Diff}(\mathbb{R}^2)$ $\exists!$ $h, k \in T_{id}\text{Diff}(\mathbb{R}^2)$

such that $(\mathbf{R}_\phi)_* h = a$ and $(\mathbf{R}_\phi)_* k = b$; where $T_{id}\text{Diff}(\mathbb{R}^2)$ is the tangent space at ϕ_{id}, and $(\mathbf{R}_\phi)_*$ is the pushforward mapping[40], where the derivative is with respect to some γ. Thus, a right-invariant metric is defined by the inner product at $T_\phi\text{Diff}(\mathbb{R}^2)$:

$$\langle a, b\rangle_\phi := \langle h, k\rangle_{id} \tag{2.13}$$

Up until now, we have described a distance metric, d^*, on $\text{Diff}(\mathbb{R}^2)$ and on the space of our shapes, but we have not discussed how one can find the deformation function $\phi \in \text{Diff}(\mathbb{R}^2)$. This brings us onto the topic of geodesics.

Geodesics

There are infinitely-many possible paths $\phi \in \text{Diff}(\mathbb{R}^2)$ that can transform our template curve to c_1. But our interest is on the shortest path, with respect to a given distance metric. We call such paths *geodesics*, and we note that they are not necessarily unique.

The ease of finding geodesics depends on the metric we define in the space. Though this can sometimes be an easier task, for example, when working with points on a sphere, a geodesic is simply the shorter arc of the great circle passing through the points; at other times, it is not so trivial, particularly, when working with an infinite-dimensional Riemannain manifold. Here, the geodesic equation is a PDE, which we shall not go into but once again recommend (Marsland and Sommer, 2020), for further details.

There are two main points of interest once an equation for a geodesic path has been established. Firstly, we may be interested in the transformation of a template curve to the target curve, via a geodesic path. This is seen in Figure 2.4, where a closed leaf outline is morphed into another. Secondly, our sole interest may be in quantifying the differences between two shapes, and thus we seek a *distance*, such as the distance discussed in Equation (2.12). Furthermore, we can either use the length of a geodesic to quantify the differences between shapes, or alternatively, we can compute the energy (which is analogous to the square of the length):

Definition 2.16 (Geodesic Energy) *Geodesics are smooth curves $\phi : [0, 1] \mapsto$ Diff($\mathcal{C}$). Their energy is defined as:*

$$E(\phi) = \frac{1}{2}\int_0^1 ||\dot{\phi}||^2_{\phi(t)} dt$$

[40]Pushforward mappings can be thought of as the *best* linear approximation to a function between smooth manifolds, which in essence, is the function's derivative.

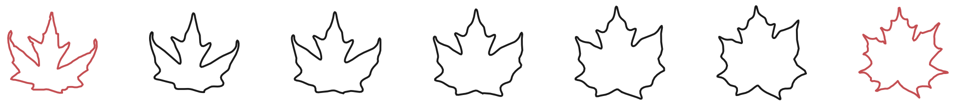


Figure 2.4: Geodesics between outlines of an ancient Miocene leaf and a close modern relative. The original template shape and target shape, are shown in red, whilst the five samples equally spaced along the geodesic path are plotted in black.

where $||\cdot||_\phi$ is the norm associated with the inner product at the tangent space T_ϕDiff$(\mathcal{C})$ defined by some right-invariant metric; where $\mathcal{C}$ is some smooth manifold.

Recall that we can define the *shape* of a curve by its equivalence class, as seen in Section 2.2.3. Henceforth, in order to describe a distance between the *shapes* of the template, c_0, and target, c_1, we search for the *shortest* geodesic, with respect to the energy in Equation (2.16), between all possible pairs $([c_0],[c_1]) \in S$, where S is the space of our shapes.

2.3.2 Finding Geodesics

In Section 2.2.3, we described a shape as a curve modulo the *shape-preserving transformations.* There, we saw that the shape space for these curves can be described as a quotient space, such as in Equation (2.11). However, throughout this section, we have kept a vagueness around the shape space we are working within. This is primarily because when working within the LDDMM framework, as we've seen, distances are computed directly on Diff$(\mathbb{R}^2)$ instead of the quotient space. This means in order for our distance metric to be invariant to the shape-preserving transformations (e.g., reparametrization and scaling), these similarity transforms need to be dealt with or *removed* beforehand, prior to any analysis focusing on *shapes.*

On the other hand, instead of finding geodesics / distances on a rather complicated infinite-dimensional Riemannian manifold such as Diff$(\mathbb{R}^2)$, we can employ a representation of the curves that will transform the space into one that is equipped with much simpler metrics. This is the motivation behind some of the alternative methods in elastic shape analysis that find a geodesic distance between shapes. There are two in particular that we are interested in, that we will be using for some of our projects in the subsequent chapters of this thesis. These are the Geometric Currents method, and a framework based on the Square Root Velocity Function (SRVF) representations of curves. We illustrate

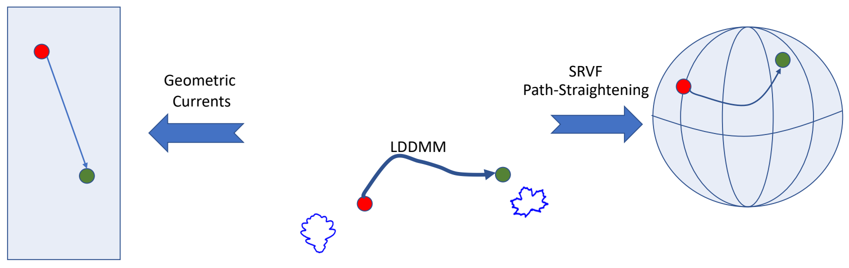


Figure 2.5: Shape space illustration of leaf outlines. LDDMM methods work on the original space, whilst the Geometric Currents and SRVF Path-Straightening methods transform the curves, and hence the pre-shape and shape spaces, into something much simpler to work on.

the ideas behind all of these methods in Figure 2.5. This Figure shows how LDDMM methods work directly on the original shape space, whilst the Geometric Currents method transforms the space so that geodesics are simply straight lines, and the SRVF framework allows us to transform the space into a sphere. Here, we will briefly go over these methods, whilst we dedicate the succeeding section of this chapter, to a more detailed overview of the SRVF framework, as it will play a large role in our research.

Solutions to the LDDMM Model

Within the LDDMM framework, it is common to employ vector-field representations of the diffeomorphisms in the space, where by such representations, we imply the following:

Let S be the original shape space of the curves. A time-dependent vector field representation of $\phi \in \text{Diff}(\mathbb{R}^2)$ can be represented as $\nu : [0,1] \times S \mapsto \mathbb{R}^2$, where we can think of the transformation as *starting* at $t = 0$, and terminating at $t = 1$. Hence, we re-write the geodesic energy in Definition (2.16), as follows:

$$E(\phi) = \int_0^1 ||v_t||^2 dt \tag{2.14}$$

Consequently, the question of finding a distance between two curves, c_0, c_1, is analogous to the optimization problem concerning the minimization of this geodesic energy <u>and</u> the distance between the resulting, transformed template, with the target:

$$d_S(c_0, c_1) = \min_{\phi \in \text{Diff}(\mathbb{R}^2)} E(\phi) + ||\phi \cdot c_0 - c_1||^2. \tag{2.15}$$

We note that the Euler-Lagrange equations can be used to minimize the vector field. For more details, see (Marsland and Sommer, 2020) and (Younes, 2010).

Geometric Currents

For two curves $c_0, c_1 \in \mathrm{Emb}(\mathbb{S}^1, \mathbb{R}^2)$, our aim is to find a distance between their shapes. To find a distance between two shapes with the LDDMM methods, we must first remove the shape-preserving transformations in the pre-processing step (for example, with Procrustes alignment). However, it is possible to instead define a distance metric to be invariant to shape-preserving transformations; including $\mathrm{Diff}(\mathbb{S}^1)$, i.e. the reparametrization group. One way to construct such a metric is with *geometric currents*. A current or *geometric current* is defined in the field of geometric measure theory as a type of linear functional. It was first introduced in (De Rham, 1973). In recent decades we have seen its potential in shape analysis, for use in defining a distance metic on a shape space, with literature including (Glaunes et al., 2008) and (Benn et al., 2019).

To outline the concept of geometric currents in shape analysis, we first consider our curves as immersions (where immersions are locally en embedding[41]), such that $c_0, c_1 \subset \mathrm{Emb}(\mathbb{S}^1, \mathbb{R}^2) \subset \mathrm{Imm}(\mathbb{S}^1, \mathbb{R}^2)$. In order to define a current, we look at the general definition presented in (Benn et al., 2019) for the space of immersions, $Imm(M, N)$, where, in our case, $M = \mathbb{S}^1$ and $N \subseteq \mathbb{R}^2$.

Definition 2.17 (Dual (*informal definition*)) *For a vector space, U, its dual, U^*, is the vector space of linear functions of U.*

Definition 2.18 (Definition 1 in (Benn et al., 2019)) *Let M and N be oriented manifolds of dimensions m and n, with $m \leq n$. We denote by $Imm(M, N)$ the space of immersion from M to N. For $\phi \in Imm(M, N)$, $[\phi]$ is defined to be a linear function on forms given by the following:*

$$[\phi] : \Lambda^m(N) \mapsto \mathbb{R}, \ [\phi](\alpha) := \int_M \phi^* \alpha = \int_{\phi(M)} \alpha, \tag{2.16}$$

where $\Lambda^m(N)$ is the space of smooth m-forms on N. The current map is defined as:

$$[\cdot] : Imm(M, N) \mapsto \Lambda^m(N)^*, \tag{2.17}$$

where $\Lambda^m(N)^*$ is the topological dual of $\Lambda^m(N)$

[41]We note that all embeddings are also immersions. However, the latter is not always true because immersions do not necessarily exhibit injectivity, unlike embeddings which inherit bijectivity (and hence injectivity) from their definition of being homeomorphisms.

It can be shown that the current map is invariant to all orientation-preserving reparametrizations (diffeomorphisms of $\mathbb{S}^1$). This can be seen in Proposition 1 of (Benn et al., 2019). Thus we can work on the shape space, $\mathcal{S}$, defined as:

$$\mathcal{S} = {Imm(\mathbb{S}^1, \mathbb{R}^2)}\big/_{\text{Diff}^+(\mathbb{S}^1)} \tag{2.18}$$

where $\text{Diff}^+(\mathbb{S}^1) \subset \text{Diff}(\mathbb{S}^1)$ is the space of all orientation-preserving reparametrizations.

This is an important consequence as it enables us to work with the shapes of the curves, $c_0, c_1 \in \text{Imm}(\mathbb{S}^1, \mathbb{R}^2)$, by mapping their equivalence classes. As we are working with $\mathbb{S}^1$, in our case $m = 1$, this means that the current mapping defined in (2.17), can be used to map the equivalence classes to the dual space of 1-forms, thus, to a single point. This is now a vector space, and the standard Euclidean metric can be used.

By linearising and hence simplifying the space, the geometric currents method allows us to find distances between shapes much more easily than other methods within elastic shape analysis, and computationally quickly. It also allows for *traditional* statistical analysis methods to be used, such as Principal Component Analysis[42] for example, as we can work within a linear space. Though there are disadvantages to this approach, such as the inability to *invert*[43] the mapping in order to see a meaningful *average shape* in this space for analysis, its advantages, such as its computation simplicity or its robustness to noise, as detailed in (Benn et al., 2019), certainly outweigh its disadvantages. For this reason, we utilize this approach in some of our projects, detailed in Chapter 4.

Square Root Velocity Framework

The final method that we will discuss is based on the representation of curves by square root velocity functions. For our curves $c_0, c_1 \in \text{Emb}(\mathbb{S}^1, \mathbb{R}^2)$, we find a distance between their shapes by defining a distance between the equivalence classes of their square root velocity representations, $[q_0], [q_1]$, respectively, where q_0, q_1 are:

$$q_0 = \frac{\dot{c_0}}{\sqrt{|\dot{c_0}|}}$$

[42]Note that as we have found representations of the shapes of our curves in a vector space, PCA can be carried out in this space. Thus, this is different from Tangent PCA, a method in Elastic Shape Analysis, which we will discuss later on in Section 2.5.3.

[43]More specifically, a computation of an average would be based on on the mapping we saw in (2.17). However, since the mapping cannot be *reversed*, any average computed cannot be exhibited in the original space of the curves, and hence, visually, it may not be very helpful.

$$q_1 = \frac{\dot{c_1}}{\sqrt{|\dot{c_1}|}}$$

As we will soon discover, the space of all such square root velocity functions (SRVFs) is analogous to $\mathbb{L}^2(\mathbb{S}^1, \mathbb{R}^2)$, thereby transforming the pre-shape space into a unit sphere. This plays a crucial role in allowing for much simpler computations of geodesics in not just the pre-shape space, but the shape space (as defined in Equation (2.11)) too.

Across all the projects featured in this thesis, we will incorporate this SRVF framework. In the next section, we dive deeper into the mathematics behind this method, with a focus on open curves. Following on, in the last section of this chapter, we include an overview of a particular algorithm within the SRVF framework, namely SRVF Path-Straightening, used to find distances between the shapes of closed curves. We will also study a widely-used algorithm in functional and shape data analysis that aims to align open curves within this SRVF framework.

2.4 Registering Curves in the SRVF Framework

Previously, we mentioned that one can transform a shape space into a much *simpler* space by utilizing the square root velocity representation of the curves. Here, we shall delve deeper into this, and next, we will discover how these functions can be used to solve the pairwise registration problem between curves in $\mathbb{R}^2$. Note that, though the computations are similar, for succinctness we will focus on open curves, and highlight Chapter 6 of (Srivastava and Klassen, 2016), for more emphasis on closed curves.

2.4.1 Motivation for Pairwise Registration

Broadly speaking, pairwise registration refers to the alignment of two curves. By minimizing some distance between two curves, we can *register* the curves to one another, such that some distance between them is minimized. And in doing so, we can bring the curves into alignment, this often results in an alignment of the peaks and valleys of the curves (since the distance between the curves is large if these do not match), as shown in Figure 2.6.

The motivation to register curves in a dataset arises with the desire to analyse the shapes of our curves, in particular, with functional data, when there is no concern for the time

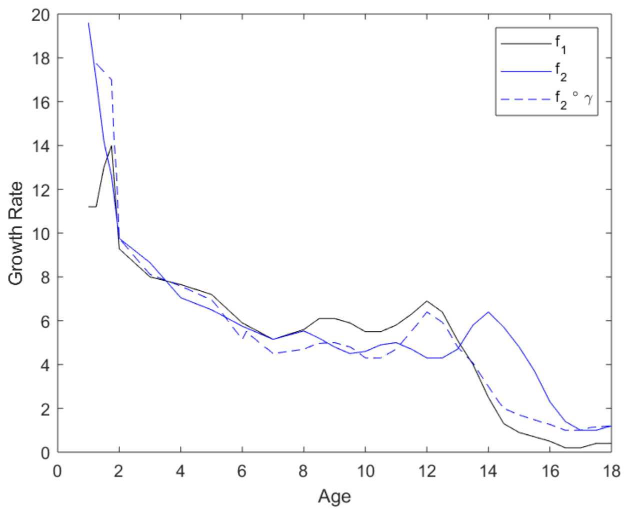


Figure 2.6: Growth rates of two people, f_1, f_2, taken from the Berkeley Growth Dataset[45], and the growth rate of the second person after registration, i.e. $f_2 \circ \gamma$, where γ is the reparameterization function found during pairwise registration of f_1 and f_2 using a dynamic programming algorithm, which we will outline in the next section.

domain involved. For example, take two curves $f_1, f_2 : I \mapsto \mathbb{R}^2$, on some fixed-time domain, I, representing the growth curves of two people, as seen in Figure 2.6. Suppose we wish to identify the average growth rate. One simple traditional way to do this would be to compute a standard mean of curves by finding the pointwise average of the two curves. However, since the growth spurts of the two people occur at different times, this hides the true way that growth happens. It may therefore be beneficial to first align the curves, with some reparametrization function, γ, before computing the average. Pairwise registration as a pre-processing step to further analysis is a common approach utilising the registration problem[44].

2.4.2 Square Root Velocity Functions

The representation of curves with square root velocity functions, for the purpose of shape analysis, was first introduced in (Srivastava et al., 2011a). In this section, we will provide an overview of square root velocity representations, in particular, of absolutely continuous curves. For more details on this framework in functional and shape data analysis, we refer the reader to (Srivastava and Klassen, 2016).

[44]We often refer to the optimization of the alignment of two curves as the registration problem.

[45]The Berkeley Growth Dataset is from a mid twentieth-century study on the heights of children over time, from ages 1-18. An R version can be found here[46], whilst a MATLAB version can be found here[47].

Definition 2.19 (Definition 4.1 in (Srivastava and Klassen, 2016))
A function $f : I \mapsto \mathbb{R}$ is absolutely continuous on I if it satisfies the following:

1. *f is differentiable almost everywhere on I,*
2. *$f(t) = f(0) + \int_0^t \dot{f}(u)du \; \forall t \in I$.*

A known remark to the definition of absolute continuity[48] on a function f, is that its derivative is Lebesgue integrable. This will come into play later in this section.

Definition 2.20 (Lebesgue integrability) *For a non-negative, measurable function f, we define its Lebesgue integral over a measure space $\mathcal{X}$, with measure μ, as:*

$$\int_{\mathcal{X}} f \; d\mu. \tag{2.19}$$

Thereby, f is defined as Lebesgue integrable if (2.19) is finite.

Definition 2.21 (Square Root Velocity Function) *For an absolute continuous curve $c : I \mapsto \mathbb{R}^2$, we define its square root velocity function as:*

$$q(t) = \begin{cases} \dfrac{\dot{c}(t)}{\sqrt{|\dot{c}(t)|}}, & \text{if } |\dot{c}(t)| \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2.20}$$

Note that we require absolute continuity, as it is stronger than simply requiring the curves to be continuous. Consequently, for these absolutely continuous curves, the square root velocity function (SRVF), can be rewritten as $q(t) = \text{sign}(\dot{c}(t))\sqrt{|\dot{c}(t)|}$, sometimes called the square root slope function. By representing curves $c : I \mapsto \mathbb{R}^2$ by their SRVF, the space of curves is significantly simplified to $\mathbb{L}^2(I, \mathbb{R}^2)$, the space of square-integrable functions on I;

Theorem 2.22 *For absolutely continuous curves, $c : I \mapsto \mathbb{R}^2$, the space of all square root velocity functions is $\mathbb{L}^2(I, \mathbb{R}^2)$.*

[48] We also remark that all Lipschitz functions are also absolute continuous but the inverse is not necessarily true.

Proof: Denote the space of square root velocity functions as Q. We show that $Q \subseteq \mathbb{L}^2(I, \mathbb{R}^2)$ and conversely, $\mathbb{L}^2(I, \mathbb{R}^2) \subseteq Q$.

We begin by showing that $Q \subseteq \mathbb{L}^2(I, \mathbb{R}^2)$. Recall the definition of square-integrability for a function y over the real line is $\int_I |y(x)|^2 dx < \infty$.

$$\int_I |q(t)|^2 dt = \int_I \left| \frac{\dot{c}(t)}{\sqrt{|\dot{c}(t)|}} \right|^2 dt = \int_I \frac{\dot{c}(t)^2}{|\dot{c}(t)|} dt = \int_I |\dot{c}(t)| dt < \infty,$$

where the last inequality follows from the definition of absolute continuity, as the derivative of c is Lebesgue integrable. This shows us that $\forall q \in Q$, q is square-integrable and thus $Q \subseteq \mathbb{L}^2(I, \mathbb{R}^2)$.

Conversely, to prove that $\mathbb{L}^2(I, \mathbb{R}^2) \subseteq Q$, we take a function $q \in \mathbb{L}^2(I, \mathbb{R}^2)$:

$$\int_I |q(t)|^2 dt < \infty.$$

This implies that there exists an absolutely continuous curve $c : I \mapsto \mathbb{R}^2$ s.t.

$$\int_I |q(t)|^2 dt = c(t).$$

By differentiating this, up to translation, we get:

$$|q(t)|^2 = \dot{c}(t) \implies |q(t)| = \sqrt{|\dot{c}(t)|} \implies q(t) = \frac{\dot{c}(t)}{\sqrt{|\dot{c}(t)|}}$$

This shows that $\forall q \in \mathbb{L}^2(I, \mathbb{R}^2)$, $q \in Q$ and thus $\mathbb{L}^2(I, \mathbb{R}^2) \subseteq Q$.

Since we have shown that $\mathbb{L}^2(I, \mathbb{R}^2) \subseteq Q$ and $Q \subseteq \mathbb{L}^2(I, \mathbb{R}^2)$, we have proved that the space of all SRVFs $Q = \mathbb{L}^2(I, \mathbb{R}^2)$. $\square$.

The fact that the space of square root velocity functions is $\mathbb{L}^2(I, \mathbb{R}^2)$ is very significant for employing SRVF representations of curves, as we are now working in a much simpler space, analogous to a unit sphere, which can be equipped with a much simpler metric to compute distances between shapes: the $\mathbb{L}^2$-metric. For two SRVFs $q_1, q_2 \in \mathbb{L}^2(I, \mathbb{R}^2)$, we can use the inner product as follows:

$$\langle q_1, q_2 \rangle_{\mathbb{L}^2} = \int_I q_1(t) q_2(t) dt \tag{2.21}$$

2.4.3 Invariance to Shape-Preserving Transformations

The important particularity that we have emphasised across the thesis is that we are not interested in the difference between the original curves per se, but in the differences between the *shapes* of curves. Thus, as we are not *removing* any shape invariability beforehand, we require the distance metric used to be invariant to shape-preserving transformations. Recall that here, the shape-preserving transformations refer to scaling, translation, rotation, and reparametrization.

By using the square root velocity representation of curves, we automatically avoid translation variability as a result of the derivative inside the function. In order to handle the scaling variability, we can require the curves to be of unit length and can simplify computations further by defining our interval as $I = [0, 1]$. This means that our pre-shape space is now a unit sphere. To remove rotation and reparameterization variability, we define a quotient space, as in Section 2.2.3. In advance of defining such a quotient space, we must first examine the action of the reparametrization group and that of the rotation group on the space of square root velocity functions.

Denote the space of reparameterizations as Γ and the space $\mathbb{L}^2([0,1],\mathbb{R}^2)$ as simply $\mathbb{L}^2$. We introduce the following mapping:

$$\begin{cases} \mathbb{L}^2 \times \Gamma \mapsto \mathbb{L}^2 \\ (q,\gamma) \mapsto (q \circ \gamma)\sqrt{\dot{\gamma}} \end{cases} \tag{2.22}$$

$\forall$ SRVF $q \in \mathbb{L}^2$ and reparameterization function, $\gamma \in \Gamma$.

Theorem 2.23 *The map in (2.22) forms a right group action of* Γ *on* $\mathbb{L}^2$*.*

Proof: Previously, we mentioned that $(\Gamma, \circ)$ is a group. Therefore, we introduce the mapping $\alpha : \mathbb{L}^2 \times \Gamma \mapsto \mathbb{L}^2$, that can be defined as a right group action if it satisfies the identity and compatibility axioms. Thus, $\forall q \in \mathbb{L}^2$:

1. $(q, \gamma_{id}) = (q \circ \gamma_{id}(t))\frac{\dot{\gamma}_{id}(t)}{\sqrt{|\dot{\gamma}_{id}(t)|}} = (q \circ \gamma_{id}(t))(1) = q(\gamma_{id}(t)) = q(t)$, where $\gamma_{id} \in \Gamma$ is the identity function and q is the SRVF of a curve $c(t)$.

2. Need to show that $\alpha(\alpha(q,\gamma_1),\gamma_2) = \alpha(q, \gamma_1 \circ \gamma_2)$ where $\gamma_1, \gamma_2 \in \Gamma$;

$$\alpha(q,\gamma_1\circ\gamma_2) = (q\circ(\gamma_1\circ\gamma_2))\sqrt{(\gamma_1\circ\gamma_2)'} = (q\circ(\gamma_1\circ\gamma_2))\sqrt{(\dot\gamma_1\circ\gamma_2)\dot\gamma_2}$$
$$= (((q\circ\gamma_1)\circ\gamma_2)\sqrt{\dot\gamma_1\circ\gamma_2})\sqrt{\dot\gamma_2} = ((q\circ\gamma_1)\sqrt{\dot\gamma_1})\circ\gamma_2)\sqrt{\dot\gamma_2}$$

which is equal to $\alpha(\alpha(q,\gamma_1),\gamma_2)$. □

The action of the rotation group can be defined by the mapping[49]:

$$\begin{cases} SO(2)\times\mathbb{L}^2 \mapsto \mathbb{L}^2 \\ (O,q(t)) = Oq(t) \end{cases} \tag{2.23}$$

Before we define a quotient space for these two transformation groups, we need to be sure that the actions of both of the groups is by isometries, in order for our shape analysis to be invariant to rotation and to reparameterization. This space is equipped with the $\mathbb{L}^2$ inner product $\langle\ ,\ \rangle_{\mathbb{L}^2}$ where $\langle q_1,q_2\rangle_{\mathbb{L}^2} = \int_0^1 q_1(t)q_2(t)dt$, for $q_1,q_2\in\mathbb{L}^2$.

Theorem 2.24 *The actions of rotation group $SO(2)$ and reparameterization group Γ on $\mathbb{L}^2$, is by isometries with respect to the $\mathbb{L}^2$-metric.*

Proof ((Srivastava et al., 2011a)): $\forall\, q_1,q_2\in\mathbb{L}^2$:

- $\langle(O,q_1(t)),(O,q_2(t))\rangle_{\mathbb{L}^2} = \langle Oq_1(t),Oq_2(t)\rangle_{\mathbb{L}^2} = \langle q_1(t),q_2(t)\rangle_{\mathbb{L}^2}$, where $O\in SO(2)$. Thus the action of $SO(2)$ is by isometry.
- For $\gamma\in\Gamma$, $\langle(q_1,\gamma),(q_2,\gamma)\rangle_{\mathbb{L}^2} = \int_0^1 q_1(\gamma(t))\sqrt{\dot\gamma}q_2(\gamma(t))\sqrt{\dot\gamma}dt$. By setting $s=\gamma(t) \implies ds = \dot\gamma(t)dt$, we obtain $\int_0^1 q_1(s)q_2(s)ds = \langle q_1,q_2\rangle_{\mathbb{L}^2}$. Therefore the action of the reparametrization group, Γ, is by isometry. □

In essence, Theorem 2.24 shows us that the distances between two SRVFs, with respect to the $\mathbb{L}^2$-metric, are not affected by the above transformations. Henceforth, we can define a joint action between the rotation and the reparameterization group on $\mathbb{L}^2$ and define the shape space, $\mathcal{S}_{SRVF}$, in this square root velocity framework as:

$$((O,\gamma),q) := O(q\circ\gamma)\sqrt{\dot\gamma} \tag{2.24}$$

[49]Note that with our curves, the transformation, $Oq(t)$, can be computed by standard multiplication between the rotation matrix and coordinates (represented as column vectors) that make up the curve.

$$\mathcal{S}_{\text{SRVF}} = {}^{\mathbb{L}^2}\!\!/_{\Gamma \times SO(2)} \tag{2.25}$$

And thus, we define our shapes in this space by:

$$[q] := \{(O,\gamma), q) \mid q \in \mathbb{L}^2,\ (\gamma, O) \in \Gamma \times SO(2)\}. \tag{2.26}$$

For more details about shape-preserving transformations, and the actions of transformations groups on the space of square root velocity functions, we refer the reader to (Srivastava et al., 2011a) and (Srivastava and Klassen, 2016).

2.4.4 Geodesics & Distances

We are interested in the shortest distance between a pair of shapes. Recall from section 2.3.1, that the length of a geodesic between two curves can represent a *distance* between them. On the pre-shape space this task is straightforward as the space is the unit sphere, and hence geodesics between points are simply the shorter arcs on great circles that connect the points. Thus, the path of a geodesic, α, connecting two curves $c_1.c_2$ with SRVFs q_1, q_2, can be computed as:

$$\alpha(\tau) = \frac{1}{\sin\theta}(\sin\theta(1-\tau)q_1 + \sin\theta\tau q_2) \tag{2.27}$$

where θ is the geodesic length and thus the distance of interest is:

$$d(q_1, q_2) = \theta = \cos^{-1}\langle q_1, q_2\rangle_{\mathbb{L}^2} \tag{2.28}$$

On the other hand, computing a distance in the shape space, $\mathcal{S}_{\text{SRVF}}$, is not so simple, as it would be between equivalence classes, not curves.

The shapes of an SRVF, q, can be described by its equivalence class $[q] \in \mathcal{S}_{\text{SRVF}}$. We can begin to formulate a distance between two shapes in the shape space $\mathcal{S}_{\text{SRVF}}$, based on the geodesic distance metric inherited from the pre-shape space, $\mathbb{L}^2$. Therefore a distance metric, $d_{\mathcal{S}}$, on $\mathcal{S}_{\text{SRVF}}$ can be defined as:

$$d_{\mathcal{S}}([q_1],[q_2]) = \inf_{\bar{q}_1 \in [q_1], \bar{q}_2 \in [q_2]} \langle \bar{q}_1, \bar{q}_2\rangle_{\mathbb{L}^2} \tag{2.29}$$

$$= \inf_{\substack{\gamma_{1,2}\in\Gamma \\ O_{1,2}\in SO(2)}} \langle O_1(q_1\circ\gamma_1)\sqrt{\dot{\gamma}_1}, O_2(q_2\circ\gamma_2)\sqrt{\dot{\gamma}_2}\rangle_{\mathbb{L}^2}. \tag{2.30}$$

Since the action of the product $\Gamma \times SO(2)$ is by isometries, we can fix the orbits of one SRVF and minimize for the other e.g.:

$$d_{\mathcal{S}}([q_1],[q_2]) = \inf_{\substack{\gamma\in\Gamma \\ O\in SO(2)}} \langle O(q_1\circ\gamma)\sqrt{\dot{\gamma}}, q_2\rangle_{\mathbb{L}^2} = \inf_{\substack{\gamma\in\Gamma \\ O\in SO(2)}} \langle q_1, O(q_2\circ\gamma)\sqrt{\dot{\gamma}}\rangle_{\mathbb{L}^2}, \tag{2.31}$$

where the last equality follows from the symmetry of $\langle\cdot,\cdot\rangle_{\mathbb{L}^2}$ and the registration of these functions by reparameterization and rotation.

2.5 Implementations in the SRV Framework

In this final section of the Background chapter, we explore various implementations of elastic shape analysis within the Square Root Velocity framework. We begin with a method of finding geodesics between closed curves in this framework using an algorithm called the SRVF Path-Straightening algorithm. Next, we explore a Dynamic Programming algorithm that is used to align open, piecewise linear curves. This algorithm finds *approximate* solutions to the Pairwise Registration problem. We also take a brief look at an alternative algorithm that provides *precise* solutions to this problem, and will subsequently detail a comparison between the two methods. Lastly, we focus on the computation of averages in the SRVF framework, via, a Karcher mean. We focus on these algorithms as they are the methods that will play the greatest parts in the projects we will subsequently explore. We note that for the implentations of these algorithms, we recommend the MATLAB package, libsrvf[50], as detailed in (Robinson, 2012), for the *approximate* and *precise* pairwise registration of open curves, and the library fdasrsf[51], for Python implementations of elastic shape analysis on open and closed curves.

2.5.1 SRVF Path-Straightening

In this subsection, we will discuss another diffeomorphic shape analysis method used to *deform* closed curves into one another: SRVF Path-Straightening. As the name suggests, for this method we use the Square Root Velocity framework, employing SRVF representations of our closed curves. Though it bears similarities to the LDDMM approach, as SRVF representations are used, the pre-shape space (and hence, shape space) of the curves is transformed into a far less complicated space, thus enabling much simpler computations to optimize curve deformations.

[50]https://github.com/fsu-ssamg/libsrvf
[51]https://fdasrsf-python.readthedocs.io/en/latest/

Pre-Shape & Shape Space

We consider unit-length closed curves, c, as embeddings, $c \in Emb(\mathbb{S}^1, \mathbb{R}^2)$. Thus, the original space of our curves, S, with an added emphasis for the closure, is defined as:

$$S = \left\{ c \in Emb(\mathbb{S}^1, \mathbb{R}^2) \mid \int_{\mathbb{S}^1} ||\dot{c}(t)||dt = 1, \int_{\mathbb{S}^1} \dot{c}(t)dt = 0 \right\}$$

Next, recall the definition of a square root velocity function, SRVF, of an open curve from subsection 2.4.2. For a closed curve, as a function of time, $c(t)$, the SRVF is:

$$q(t) = \frac{\dot{c}(t)}{\sqrt{||\dot{c}(t)||}} \tag{2.32}$$

where $||\cdot||$ is the standard Euclidean norm.

Note that the unit-length and closure conditions for SRVFs are:

- **Unit-Length**: $\int_{\mathbb{S}^1} ||\dot{c}(t)||dt = 1 \implies \int_{\mathbb{S}^1} ||q(t)||^2 dt = 1$
- **Closure**: $\int_{\mathbb{S}^1} \dot{c}(t)dt = 0 \implies \int_{\mathbb{S}^1} q(t)||q(t)||dt = 0$ as the ratio $\frac{q(t)}{||q(t)||}$ is the instantaneous direction along $c(t)$, (Joshi et al., 2007).

If we restrict the curves to be absolutely continuous (see Definition 2.19) on $\mathbb{S}^1$, then for a curve $c \in Emb(\mathbb{S}^1, \mathbb{R}^2)$, its square root velocity function $q \in \mathbb{L}^2(\mathbb{S}^1, \mathbb{R}^2)$. Henceforth, we can describe our pre-shape space, S, as:

$$S = \{q \in \mathbb{L}^2(\mathbb{S}^1, \mathbb{R}^2) \mid \int_{\mathbb{S}^1} ||q(t)||^2 dt = 1, \int_{\mathbb{S}^1} q(t)||q(t)||dt = 0\} \tag{2.33}$$

By ignoring the closure condition in (2.33), we get a space, say S_o, that is geometrically equivalent to a sphere, similar to the pre-shape space definition in Section 2.4.2. Thus, as described earlier, our space of unit length curves has been transformed into a subset of the unit-sphere. Moreover, S is a submanifold, with $S \subset S_o \subset \mathbb{L}^2(\mathbb{S}^1, \mathbb{R}^2)$, as shown in (Joshi et al., 2007).

Now that the pre-shape space has been defined, we can begin to describe the shape-space. As seen in section 2.2.3, we wish to define the *shape* of a curve modulo shape-preserving transformations and reparameterizations. As scaling and translation invariance are already handled in the pre-shape space, S, all that remains is rotation and reparameterization. Recall from section 2.2.2, that for curves $c \in Emb(\mathbb{S}^1, \mathbb{R}^2)$, the set of reparametrizations is the diffeomorphism group $\text{Diff}(\mathbb{S}^1)$. Therefore, as we did in Equation (2.11), we can define the shape space, $\mathcal{S}$, as the quotient space of our pre-shape space, modulo the rotation and reparameterization group:

$$\mathcal{S} = {}^{S}\!\big/_{SO(2) \times \mathrm{Diff}(\mathbb{S}^1)}. \tag{2.34}$$

Path-Straightening

Our primary aim is to deform one closed curve by the shortest path into another in order to describe a distance between their shapes. Naturally, this brings us back to the topic of geodesics. In this section, we follow the methods described in (Joshi et al., 2007), which employ a path-straightening algorithm to find geodesics between curves.

As seen previously, the formulation of geodesics depends on the chosen Riemannian metric on our pre-shape space, or, alternatively, on the inner-product on the tangent space of the pre-shape space. One advantage of our pre-shape space is that we can use the standard $\mathbb{L}^2$-metric. We combine this with a definition of a normal space, as in (Srivastava et al., 2011a), to define a tangent space to S at an SRVF, q, represented as a discretization based on n points:

$$T_qS := \{w : \mathbb{S}^1 \mapsto \mathbb{R}^n \mid w \in T_q(S_o), w \perp N_q(A)\} \tag{2.35}$$

where A is the space of closed curves, $S = S_o \cap A$, and $N_q(S)$ is the normal space, as defined in the aforementioned literature:

$$N_q(S) = \operatorname{span}\left\{ q(t), \left(\frac{q(t)_i}{||q(t)||} q(t) + ||q(t)||\mathbf{e}_i \right)_{i=1,\cdots,n} \right\} \tag{2.36}$$

with $\mathbf{e}_i$ as the i^{th} column of the identity matrix, I_n.

We remark that equation (2.35) shows us that the tangent vectors at S, lie on the tangent space to S_o i.e., the space defined in (2.33), which excludes the additional closure condition. We will shortly return to this point.

For now, we consider two closed curves c_1, c_2 with SRVFs $q_1, q_2 \in S$. Let $\alpha : [0, 1] \mapsto S$ be a path from q_1 to q_2, where $\alpha(0) = q_1$ and $\alpha(1) = q_2$. We let $\mathcal{H}$ denote the set of all paths in S and $\mathcal{H}_0 \subset \mathcal{H}$ denote the set of paths from q_1 to q_2. To find geodesics between q_1 and q_2, the Path-Straightening method, (Joshi et al., 2007), involves working on the tangent space of $\mathcal{H}_0$ at paths such as α. We define a tangent space, $T_\alpha(\mathcal{H}_0)$, at the path α, to $\mathcal{H}_0$, as follows:

$$T_{\alpha(\tau)}(S) = \{w \mid w \perp N_{\alpha(\tau)}(S)\} \tag{2.37}$$

$$T_\alpha(\mathcal{H}) = \{w \mid \forall \tau \in [0,1],\ w(\tau) \in T_{\alpha(t)}(S)\} \tag{2.38}$$

$$T_\alpha(\mathcal{H}_0) = \{w \in T_\alpha(\mathcal{H}) \mid w(0) = w(1) = 0\} \tag{2.39}$$

where (2.39) fixes the end points of the path.

The general objective of the SRVF Path-Straightening method is to straighten a path, α, between two square root velocity functions using a gradient descent approach, until the path is a geodesic, i.e. until an energy function, E, describing a deformation, is locally minimized in $\mathcal{H}_0$. Recall the definition of the geodesic energy equation, from Definition 2.16. We can use[52] this to define an energy:

$$E(\alpha) = \frac{1}{2}\int_0^1 \left\langle \frac{d\alpha}{d\tau}, \frac{d\alpha}{d\tau} \right\rangle_\alpha d\tau, \tag{2.40}$$

where $\langle\cdot,\cdot\rangle_\alpha$ is the inner product defined on the tangent space, $T_\alpha(\mathcal{H}_0)$.

The pre-shape space, S, is a much simpler space to work on than the shape space, $\mathcal{S}$, particularly because we can employ the standard $\mathbb{L}^2$-metric. Therefore, it is computationally more efficient to find geodesics between curves in the pre-shape space first, before handling the shape-preserving transformations that turn that into the shape space.

Recall that the shape space $\mathcal{S}$ is the quotient space of S modulo the rotation group, $SO(2)$, and reparameterization group, Γ. We consider two curves c_1, c_2 with SRVFs $q_1, q_2 \in S$, and we define the joint action of $SO(2) \times \Gamma$ as we did in equation (2.24), for a rotation $O \in SO(2)$ and reparameterization $\gamma \in \Gamma$. To remove rotation and reparameterization transformations, we minimize a cost function $\hat{E}$ with respect to the distance metric on the pre-shape space, S:

$$\hat{E}(O,\gamma) = d_S(q_1, O(q_2 \circ \gamma)\sqrt{\dot{\gamma}}). \tag{2.41}$$

Before outlining an algorithm incorporating SRVF Path-Straightening, we allude to the remark made earlier regarding the reliance of S_o (recall that this space is simply

[52]Note that here, we rewrite the $\dot{\phi}$ in 2.16 with $\dot{\phi} := \frac{d\alpha}{d\tau}$, and use the inner product definition.

the pre-shape space, S, in Equation (2.33) excluding the closure condition, such that $S \subset S_o \subset \mathbb{L}^2(\mathbb{S}^1, \mathbb{R}^2))$ in defining tangent spaces to S. In fact, the space S_o plays a much greater role than may be expected in the SRVF Path-Straightening algorithms. As discussed earlier in this section, and in Section 2.4.2, the pre-shape space when working within the SRVF framework on open curves is the unit sphere. Whilst $S \subset S_o$, the computation of geodesics in S_o is much more straightforward. For this reason, it is often the case that the path derivatives, $\frac{d\alpha}{d\tau}$, as seen in the energy equation (2.40), are computed on S_o, followed by a *projection* onto S. For more details of the computations of such projections, we refer the reader to (Joshi et al., 2007).

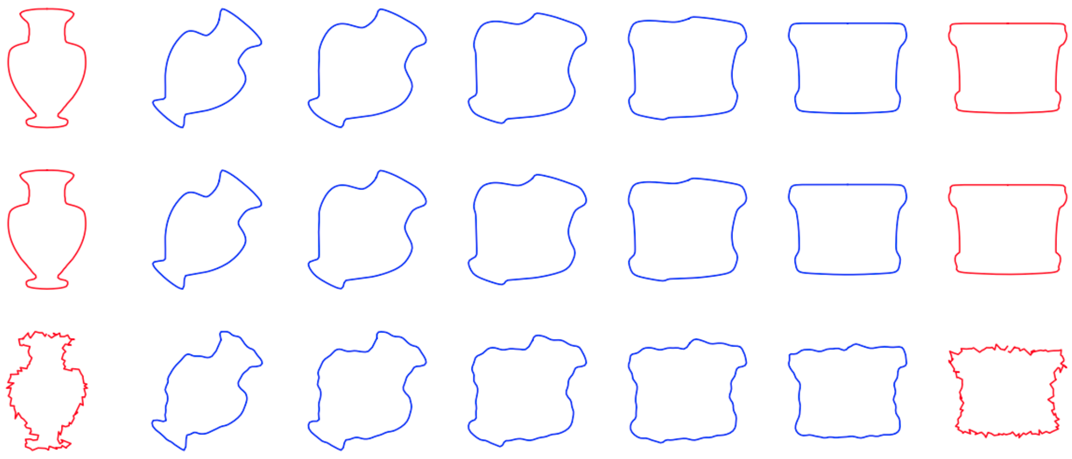


Figure 2.7: Geodesics between an amphora (left, red) and a pyxis vase (right, red). We used SRVF Path-Straightening to find a geodesic between two closed outlines. At five equally spaced time-steps, we composed the curve in the original space, with the resulting path, α, to visualize the deformations over time (seen in blue) when morphing one curve into the other. In the first row we have the original amphora and pyxis curve, and their deformations. In the second row, we have used procrustes alignment as a pre-processing step before implementing the SRVF Path-Straightening algorithm. As the Procrustes alignment did not visibly alter the curves, their deformations remain the same. In order to see the possible effect of noise, in the third row we have added Gaussian noise to the original amphora and pyxis vase outlines. As we can see, the deformations in this row are similar to the deformations seen in the other rows, except the curves are naturally less smooth.

We will now crudely summarize the SRVF Path-Straightening algorithms which aims to find a geodesic between *shapes* of closed curves. The first step is to find a geodesic between SRVF representations of the curves, in the pre-shape space. To do this, a gradient descent approach is taken to minimize the energy, $E(\alpha)$ (Equation (2.40)), for an arbitrary path,

α, until it is a geodesic. In general, this can be followed by the removal of rotation and reparameterization transformations, which are done separately, using methods described in (Srivastava et al., 2011a). Subsequently, a cost function, $\hat{E}$ (Equation (2.41)), is computed and compared to a threshold; if it's small enough, the algorithm stops; otherwise, it will repeat the previous step. An example of a curve being deformed into another curve, using a geodesic found with a SRVF Path-Straightening algorithm is shown in Figure 2.7. We note that for a more concise outline of an SRVF Path-Straightening algorithm we recommend the literature: (Klassen and Srivastava, 2006), (Joshi et al., 2007), and (Srivastava et al., 2011a).

SRVF Path-Straightening methods are very useful for the analysis of shapes of closed curves. The computations of geodesics in this framework are much simpler, and algorithmically more cost-effective than the LDDMM framework discussed in Section 2.3.1. Additionally, with SRVF Path-Straightening, geodesic distances can be computed from the paths[53], which can provide a quantification of differences between two shapes – something which we will utilize in our applications in Chapter 4. We note that this method does not offer an uncertainty in the results, and to deal with this, we would need a stochastic differential equation formulation, which is outside the scope of this thesis, but well discussed in (Marsland and Shardlow, 2017), a study on image registration with uncertainty. Moreover, on the topic of *noise*, the SRVF Path-Straightening algorithm has shown to perform well despite introductions of noise, as illustrated in the final row of Figure 2.7. This is not just the case with the visual comparisons of the deformations over time, but with the overall distances too – in this example, the geodesic distance between the original curves was computed to be 0.57, and for the curves with added Gaussian noise, this was 0.55. For all the benefits that SRVF Path-Straightening methods presents, we made the decision to incorporate these methods in our projects, which we will discover further afield.

2.5.2 Pairwise Registration of Open Curves

The alignment of open curves is the primary aim in this section. For now, we put rotations aside, and since we are working with SRVFs, all that remains to focus on is reparameterizations. In order to best align two curves, we search for a reparameterization function

[53]For example, by computing the sum of the distances between the SRVFs of consecutive deformations, with the equation seen in (2.28).

that optimally aligns curves to one another - this is known as the Pairwise Registration Problem. Previously, in Section 2.3.1, we discussed the role of diffeomorphisms in morphing and aligning curves together. Here, we specify the set of these reparameterizations, often called the set of warping functions, as follows:

$$\Gamma = \{\gamma : [0,1] \mapsto [0,1] \big| \, \gamma(0) = 0, \gamma(1) = 1, \dot{\gamma} \geq 0 \text{ almost everywhere.}\} \tag{2.42}$$

More precisely, for the algorithms we describe in this section, we require the warping functions $\gamma \in \Gamma$ to be increasing. Our goal is to find *optimal* warping functions, γ_1^*, γ_2^* such that the square root velocity functions q_1, q_2 of curves c_1, c_2 are at their closest to each other within their orbits. In other words, $(q_1, \gamma_1^*) \in [q_1]$, $(q_2, \gamma_2^*) \in [q_2]$ such that:

$$\gamma_1^*, \gamma_2^* = \arg \inf_{\gamma_1, \gamma_2 \in \Gamma} d_{\mathcal{S}}(c_1 \circ \gamma_1, c_2 \circ \gamma_2) \tag{2.43}$$

$$= \arg \inf_{\gamma_1, \gamma_2 \in \Gamma} d_{\mathcal{S}}((q_1, \gamma_1), (q_2, \gamma_2)) \tag{2.44}$$

$$= \arg \inf_{\gamma_1, \gamma_2 \in \Gamma} \langle (q_1 \circ \gamma_1)\sqrt{\dot{\gamma}_1}, (q_2 \circ \gamma_2)\sqrt{\dot{\gamma}_2} \rangle_{\mathbb{L}^2} \tag{2.45}$$

These optimal warping functions allow us to *optimally match* our curves c_1, c_2. Such optimal matching always exist for C^1 curves, (Bruveris, 2016). We examine two methods to find them: a dynamic programming approach that approximates a solution, and an algorithm that finds a precise solution, described in (Lahiri et al., 2015). In the following section, we apply these two methods to a commonly used dataset in order to analyse the effects of approximate solutions to the pairwise registration problem.

Dynamic Programming Algorithm

For years, a dynamic programming algorithm has been a favoured approach to solving a pairwise registration problem in shape analysis, as it is computationally efficient. We briefly go over one version of this algorithm, which we shall call DPA. For more details, we refer the reader to the appendices found in (Srivastava and Klassen, 2016).

To minimize a distance between two open, absolutely continuous curves c_1, c_2 with square root velocity functions q_1, q_2, we define a cost function, E, as the standard $\mathbb{L}^2$ norm between the SRVFs:

$$E[c_1, c_2] = ||q_1 - q_2||_{\mathbb{L}^2}. \tag{2.46}$$

As seen in (2.31), we can fix the orbits of the SRVF of one curve, say c_1, and thus we can redefine our *optimal reparameterization* as:

$$\gamma^* = \arg\min_{\gamma\in\Gamma} E[c_1, c_2 \circ \gamma] = \arg\inf_{\gamma\in\Gamma} ||q_1 - (q_2, \gamma)||_{\mathbb{L}^2}. \tag{2.47}$$

We take two piecewise linear curves c_1, c_2 with N and M points respectively, with SRVFs q_1, q_2. The initial step of DPA is to create a $N \times M$ grid labelled from $[0, 1]$ on both axes. Thus, the aim is to find the *optimal* warping function γ^* that starts at point $(0, 0)$ on the grid and ends at $(1, 1)$. Subsequently, we can define the energy $H_{i,j}$, at any $(\frac{i}{N}, \frac{j}{M})$ point at the grid with a recurrence relation, as follows:

$$H_{i,j} = \min_{(\frac{k}{N},\frac{l}{M})\in\mathcal{N}_{i,j}} H_{k,l} + E[c_1, c_2 \circ \gamma] \tag{2.48}$$

$$= \min_{(\frac{k}{N},\frac{l}{N})\in\mathcal{N}_{i,j}} H_{k,l} + \int_{\frac{k}{N}}^{\frac{l}{N}} (q_1(t) - q_2(\gamma(t))\sqrt{\dot{\gamma}(t)})^2 dt, \tag{2.49}$$

where γ is simply the straight line between points $(\frac{k}{m}, \frac{l}{m})$ and $(\frac{i}{m}, \frac{j}{m})$, and $\mathcal{N}_{i,j}$ is the neighbourhood around the point $(\frac{i}{N}, \frac{j}{M})$. This energy term is then used to create a strictly-increasing path from $(0, 0)$ to $(1, 1)$, which is our optimal warping function, γ^*.

Note that the neighbourhood, $\mathcal{N}_{i,j}$, can be written as:

$$\mathcal{N}_{i,j} = \left\{ \left(\frac{i-p}{N}, \frac{j-q}{M} \right) |p, q \in \mathbb{N}, 1 \leq p, q \leq \kappa \right\} \tag{2.50}$$

for some boundary κ. This implies that the neighbouring functions are not only bounded by some integer κ, but they are strictly south-west of the point. Furthermore, neighbouring points connected by a horizontal line are not permitted, as we require a strictly increasing γ^*. Meanwhile, neighbouring points connected by vertical lines would cause an error in the energy term.

An outline of the DPA code is presented in Algorithm 1. As seen here, in addition to the neighbourhood restrictions, and the strictly-increasing property enforced on γ^*, we see a further assumption that the reparameterization only changes slope at grid points. For all of these reasons, the resulting optimal warping function is seen as an *approximate* solution. We now take a brief look at a more recent algorithm, which finds *precise* solutions to the pairwise registration problem, which we call *KLR*.

Algorithm 1: DPA

```
DPA($c_1$,$c_2$)

Aim:
Finds optimal reparameterization between curves $c_1, c_2$.
Initial Step:
Compute square root velocity functions $q_1, q_2$ for $c_1, c_2$ respectively.
Code:
    1. Create a uniform $N \times M$ grid, from 0 to 1, where $N, M$ are the number of points
       in $c_1, c_2$ respectively.
    2. Set $H_{0,0} = 0$, $H_{:,0} = H_{0,:} = \infty$.
    3. Compute the energy, $H$, at all of the remaining points on the grid.
    4. Find strictly increasing $\gamma^*$:
        (i) Start from point $(1, 1)$ and connect it to the neighbouring point $(\frac{k}{N}, \frac{l}{M})$ that
            resulted in the minimum energy.
        (ii) Move onto the new point and repeat process until it reaches point $(0, 0)$.
        (iii) The resulting curve is the optimal warping function $\gamma^*$.

return $\gamma^*$ - optimal warping function aligning $c_2$ to $c_1$.
```

Precise Algorithm - KLR

The KLR algorithm, named after the authors Eric Klassen, Sayani Lahiri and Daniel Robinson, who introduced it in (Lahiri et al., 2015), is a method of finding optimal repamaterizations within the SRVF framework. For pairs of open, piecewise-linear curves in $\mathbb{R}^2$, the algorithm produces a precise solution to the pairwise registration problem, which *optimally* aligns the pairs together. We provide a brief outline of this algorithm here.

For two piecewise linear curves $c_1, c_2 : [0, 1] \mapsto \mathbb{R}^2$ with square root velocity functions q_1, q_2, the KLR algorithm searches for an optimal reparameterization, (which we'll call α^*, so as to not confuse it with the γ^* found in DPA), such that the distance between the orbits $[q_1]$ and $[q_2]$ is minimised when reparameterized with $\alpha^* = (\alpha_1^*, \alpha_2^*)$. Note that here, $\alpha^* \in \hat{\Gamma}$, where $\hat{\Gamma}$ is the closure of Γ, with an equivalent action on the right by composition. Unlike γ^* in DPA, α^* is weakly increasing.

$$\alpha^* = \arg \inf_{\alpha_1,\alpha_2 \in \hat{\Gamma}} ||(q_1, \alpha_1) - (q_2, \alpha_2)||_{\mathbb{L}^2} \tag{2.51}$$

Let's consider that our curves c_1, c_2 are subdivided at $s = \{s_0 = 0, s_1, \cdots, s_m = 1\}$ and $t = \{t_0 = 0, t_1, \cdots, t_n = 1\}$ respectively. Akin to DPA, the initial step of the KLR algorithm is the creation of a grid from $(0, 0)$ to $(1, 1)$ with s and t as the y and x axes. An $n \times m$ weight matrix, W is then defined, where for each (i, j) on the grid, we have:

$$W_{i,j} = \langle u_i, v_j \rangle_{\mathbb{L}^2} \tag{2.52}$$

where u, v are the segments $u_i = q_1((s_{i-1}, s_i))$ and $v_j = q_2((t_{j-1}, t_j))$.

Where the KLR algorithm differs greatly from DPA is in the composition of the optimal reparameterization. Here, reparameterizations, α, are made up of two different segments, named P-segments and N-segments. Broadly, these can be divided into segments that start/finish at grid points and only travel via the grid lines; and segments that travel through the grid lines. An example of these segments can be seen in Figure 2.8.

Definition 2.25 (**P-Segment**) *For a reparameterization $\alpha \in \hat{\Gamma}$, a P-segment, $\alpha|_{[a,b]\subset[0,1]}$, is an injective, piecewise linear path such that:*

- *$\alpha(a) = (s_{i_0-1}, t_{j_0-1})$ and $\alpha(b) = (s_{i_1-1}, t_{j_1-1})$ are vertices, for $i_0 \leq i_1$ and $j_0 \leq j_1$, but $\forall z \in (a, b)$, $\alpha(z)$ is not a vertex.*
- *$W_{i_{0,1}, j_{0,1}} > 0$*

Definition 2.26 (**N-Segment**) *For a reparameterization function $\alpha \in \hat{\Gamma}$, an N-segment, $\alpha|_{[a,b]\subset[0,1]}$, is a segment of α such that:*

- *$\alpha(a) = (s_{i_0}, t_{j_0})$ and $\alpha(b) = (s_{i_1}, t_{j_1})$ are vertices with $i_0 \leq i_1; j_0 \leq j_1$.*
- *$\alpha|_{[a,\frac{a+b}{2}]}$ is a horizontal, linear path from (s_{i_0}, t_{j_0}) to (s_{i_1}, t_{j_0}). And $\alpha|_{[\frac{a+b}{2},b]}$ is a vertical, linear path from (s_{i_1}, t_{j_0}) to (s_{i_1}, t_{j_1}). If $i_0 = i_1$ or $j_0 = j_1$, then the segment is a vertical or horizontal line respectively.*

For a proof that the partitions of $\alpha \in \hat{\Gamma}$, $\alpha|_{[a,b]}$, where $[a, b] \subset [0, 1]$, are either P-segments or N-segments, and for a more detailed definition of these segments, we refer the reader to (Lahiri et al., 2015).

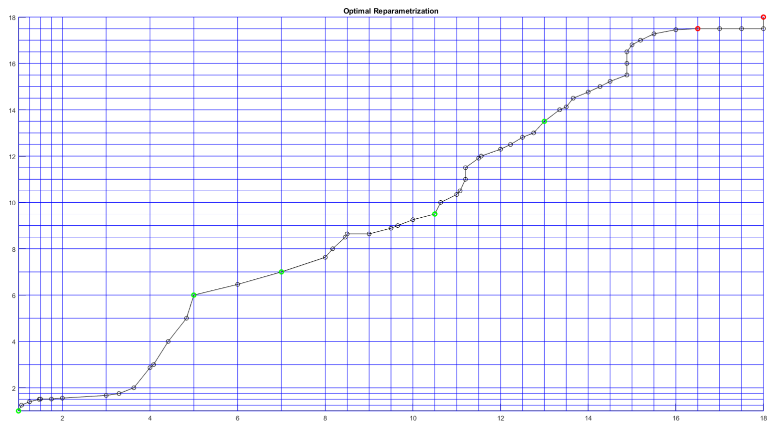


Figure 2.8: Here, we see an example of α^* on a grid, obtained by aligning two functions from the Berkeley Growth dataset. Green markers represent start and end-points of the optimal P-segments, whilst red markers represent start and end points of the optimal N-segments.

Minimizing the distance between two points on the grid is equivalent to maximizing their inner product. Thus, in the KLR algorithm, a cost function, $A_{i,j}$, is based on the inner product between the reparameterized SRVFs, which describes the energy of joining (s_i, t_j) to a segment from (s_0, t_0).

In Algorithm 2, we provide a brief overview of the KLR algorithm. We note that the KLR method is not the sole algorithm designed to provide precise solutions to the pairwise registration problem of piece-wise linear, open curves. An earlier method was in fact introduced by Daniel Robinson in (Robinson, 2012).

Precise vs Approximate Solutions

We have just outlined two methods used to solve the pairwise registration problem: DPA and KLR. These methods find optimal reparameterizations, also known as optimal warping functions, that align, or *register* two open curves. An example of such registration between two curves using these methods can be seen in Figure 2.9. Subsequently, the optimal reparameterization, can be used to find a distance between the shapes of the two curves, by computing the $\mathbb{L}^2$-distance between the optimally reparameterized square root velocity functions of the curves.

Whilst DPA produces an approximate solution to the pairwise registration problem, the KLR algorithm finds a precise optimal matching between curves, albeit at a computa-

Algorithm 2: KLR

`KLR(`c_1`,`c_2`)`

Aim:

Finds optimal reparameterization between curves c_1, c_2.

Initial Step:

Compute square root velocity functions q_1, q_2 for c_1, c_2 respectively.

Code:

1. Create a uniform $N \times M$ grid, from 0 to 1, where N, M are the number of points in c_1, c_2 respectively. The axes are the subdivisions of c_1, c_2; $s = \{s_0 = 0, s_1, \cdots, s_m = 1\}$, $t = \{t_0 = 0, t_1, \cdots, t_n = 1\}$.

2. Compute the weight matrix, W.

3. Find the optimal segment, α^*, from (s_0, t_0) to (s_m, t_n):

```
for $i \in 0, \cdots, m$ do
    for $j \in 0, \cdots, n$ do
        if segment has been found from $(s_0, t_0)$ to $(s_i, t_j)$ then
            • If $W_{i+1,j+1} \leq 0$: Look for all possible $N$-segments
              from $(s_i, t_j)$. Suppose $\exists$ $N$-segment ending at
              $(s_k, t_l)$. If $A_{i,j} > A_{k,l}$, then the union of the two
              segments (i.e. from $(s_0, t_0)$ to $(s_k, t_l)$ via $(s_i, t_j)$)
              yields a new possible optimal segment to $(s_k, t_l)$.
            • If $W_{i+1,j+1} > 0$: Look for all possible $P$-segments.
              Suppose $\exists$ $P$-segment at $(s_k, t_l)$. Let $A$ denote
              the cost of reaching $(s_k, t_l)$ from $(s_i, t_j)$. If $A +
              A_{i,j} > A_{k,l}$, then the union of the segments is a
              new possible optimal segment to $(s_k, t_l)$.
        else
            Move onto next vertex.
        end
    end
end
```

`return` α^* - optimal warping function aligning c_1 to c_2 together.

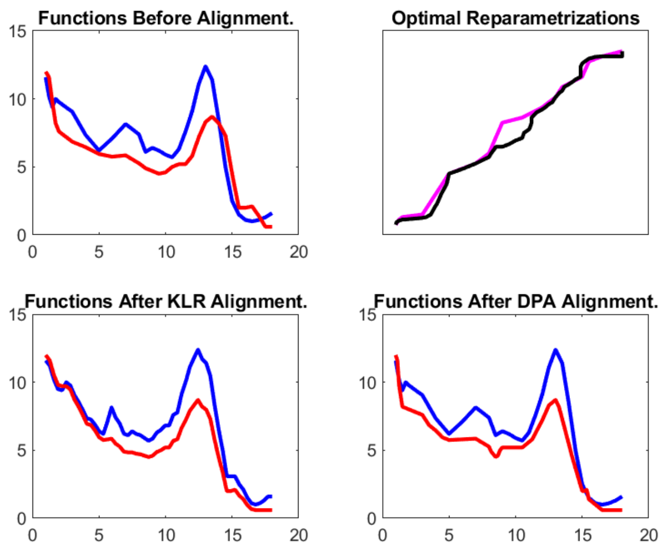


Figure 2.9: Pairwise Registration between two growth curves c_1, c_2 from the Berkeley Growth Dataset, using the KLR method and the DPA method. Top: L – original growth curves; R – γ^* from DPA (magenta), α^* from KLR (black). Bottom: L – KLR results, $c_1 \circ \alpha_1^*$ and $c_2 \circ \alpha_2^*$; R – DPA results, c_1 and $c_2 \circ \gamma^*$. Here we see that the KLR algorithm has aligned the two curves more successfully in the first half of the curves. Whilst the alignment from DPA is not too dissimilar, one noticeable difference is the smoothness of the aligned curves, when employing this algorithm, in comparison to the KLR algorithm.

tional cost. DPA remains a popular choice for aligning curves, due to its computational simplicity[54]. This motivates a study comparing the results of the two methods when applied on real data. For this, we incorporate a dataset of growth curves, the Berkeley Growth Dataset.

Recall that the optimal reparameterization found using the DPA method is entirely reliant on the grid formed, as it is assumed that the warping function only changes slope at the grid points. Therefore, we ask ourselves: *how does varying the grid-size in DPA affect the results of the pairwise registration problem?* And moreover, *how does this to compare to the KLR algorithm?* To tackle these questions, we took a sample of 54 female and 39 male height growth rate curves (aged 1-18 years) from our growth dataset and linearly interpolated the curves to have n points. We chose $n = 31,\ 69,\ 105,\ 205$, where $n = 31$ was the original number of points on the growth curves. As we're working with temporal data, the values of the chosen n were based on common growth analysis intervals e.g. $n = 69, 205$ equate to quarterly and monthly intervals respectively. Note that the original grid-size variable, $n = 31$ has non-uniform time-steps. This is because there was an added focus on the growth development in the earlier years of those in the sample. We note that the $n = 105$ grid-size is the only other *non-uniform* case. However, we note that in our experiments here, *how* we interpolate the curve is not as important, since our focus is on *how many* points we have interpolated the curves.

We begin by comparing the optimal reparameterizations, γ^*, α^* found using the DPA and KLR method respectively. Figure 2.10 shows optimal reparameterizations from the pairwise registration of 20 pairs on the left, whilst the right shows the point-wise differences between γ^* and α^*. We display the graphs for two grid-sizes, $n = 31$ and $n = 205$. Interestingly, when taking a closer look at the optimal reparameterizations, we see that the differences in γ^* as n increases are not very visible, particularly in the first half of the left-side plots. However differences in the second half do seem to appear, with γ^* and α^* looking marginally more alike when $n = 205$ compared to $n = 31$. This suggests that the optimal warping functions obtained via DPA and KLR algorithms are more alike as n is increases.

[54]For example, for two piecewise-linear curves c_1, c_2 with n and m points respectively, the processing time for DPA is $\mathcal{O}(nm\kappa)$ where κ is the neighbour boundary, as seen in (2.50); this compares to approximately $\mathcal{O}(n^2m^2)$ for the KLR method.

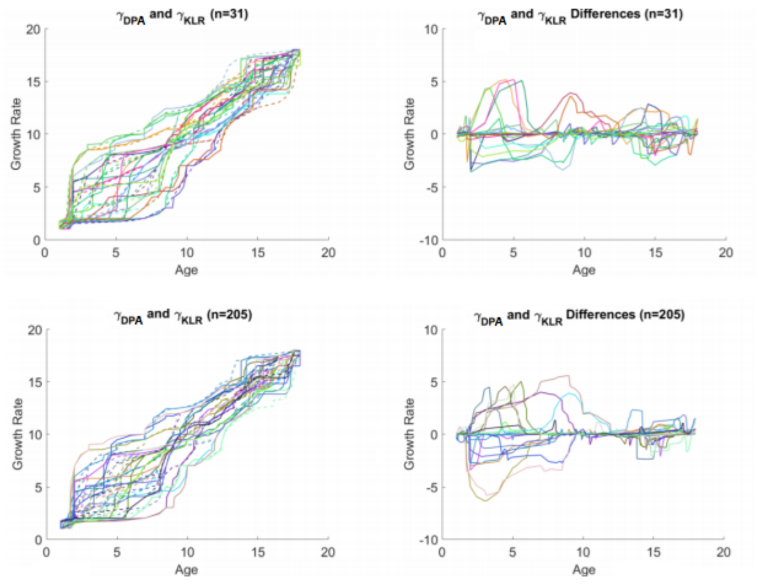


Figure 2.10: Optimal reparameterizations, γ^*, α^* resulting from pairwise registration of 20 pairs of growth curves, using DPA (line) and KLR (dotted) respectively. On the left we see the optimal reparameterizations whilst on the right we see the point-wise differences between γ^* and α^*. Moreover, in order to compare the grid-size changes, in the top row, the grid-size variable is set $n = 31$, whilst on the bottom row $n = 205$.

To study the effect of an approximate solution to the pairwise registration problem on further analyses, we computed distances using the optimal reparameterizations found with DPA and KLR, whilst varying the grid-size, n. Figure 2.11 shows the $\mathbb{L}^2$-distance after pairwise alignment of 93 curves from the Berkeley Growth Dataset, using KLR and DPA, which can be seen on the x-axis and the y-axis respectively. The closer the distances between the two methods are, the more the scatter plot should lie along the line $y = x$. Firstly, the graphs show that the DPA distances are greater than the KLR distances. Additionally, the scatter plots show slightly more resemblance to the line $y = x$ as n increases. These results are also reflected in Figure 2.12, where we employ histograms to display the differences between the distances from the two methods. Lastly, Table 2.1 displays the average *distance error*, DE, i.e., the average of the absolute differences between distances computed on pairs of optimized curves using the DPA and KLR method. The results in the table echo previous results implying a slight decrease in the differences between distances from DPA and KLR as the grid-size, n, increases.

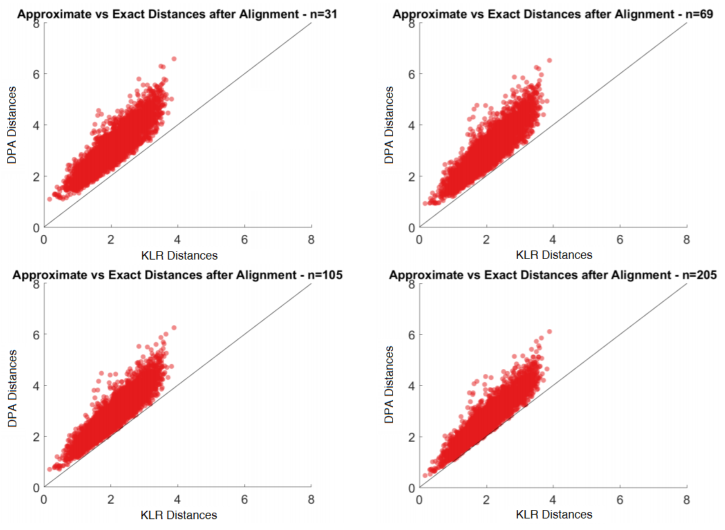


Figure 2.11: $\mathbb{L}^2$-distances between pairs of growth curves that have been aligned using the (approximate) DPA method and the (exact) KLR method. We interpolate the curves at n points, where $n = 31, 69, 105, 205$, before employing the pairwise registration algorithms. If the resulting distances using DPA and KLR were equal, then the scatter plots would resemble $y = x$ (drawn here with a black line). The graphs show that as n increases, the distances become more similar, and the scatter plots tend slightly more towards the line $y = x$. This remark is backed up by regression analysis performed on the results, which found R^2 to gradually increase from 0.80 for $n = 31$, to $R^2 = 0.86$ for $n = 205$, whilst the standard error (SE) steadily decreased from $SE = 0.37$ to $SE = 0.30$.

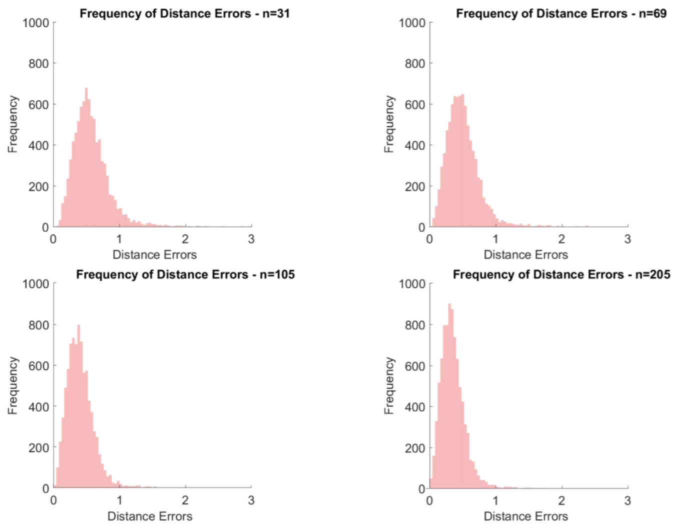


Figure 2.12: Histograms of differences in $\mathbb{L}^2$-distances using the DPA and KLR method. As the KLR method provides an exact matching between two curves, we define a distance error as the difference between the resulting distance with DPA, with the *ground truth* resulting from KLR. Note that we have used the same axes across all four plots. Though this may create a slight illusion that some of the histograms are more heavily skewed to the left than they actually are, it provides a fairer way of visualizing the comparisons between all four distance errors. Notwithstanding, the leftward skews of all four histograms indicate that in most cases the DPA and KLR distances are highly comparable. Furthermore, we can see that as the grid-size variable, n, increases, the *mode* of the histograms are more leftward, which implies a decrease in the distance errors.

DE/n	31	69	105	205
Mean	1.029	0.915	0.749	0.648
Variance	0.149	0.168	0.131	0.106
Standard Deviation	0.386	0.409	0.362	0.326

Table 2.1: Table containing the mean distance error, DE, when comparing distances resulting from DPA pairwise registration and from KLR pairwise registration. We vary the grid-size, n, in DPA, with $n = 31, 69, 105, 205$. We see that as n increases, the differences between the two methods slightly decreases.

Our tests indicate a not-too-surprising outcome that distances resulting from DPA become more similar to the results obtained via the KLR algorithm, when the gridsize, n, increases. What remains to be seen is how these results can affect further statistical analysis, for example, a Karcher mean (which we shall define in the next section). Additionally, in future tests, we will examine whether a non-linear interpolation of the curves (for the grid-sizes) can vary the results.

While the KLR algorithm may produce *precise* results, its disadvantage is with its computation complexity. The DPA algorithm is much faster[55] and as we've seen here, is capable of producing comparable results. For this reason, when it comes to the practical applications of open curve registration, we will stick to the DPA algorithm.

2.5.3 Averages in the Shape Space

Arguably, one of the most useful statistical tools in data analysis is the computation of an *average*. One such average is particularly interesting to us and that is the mean average. And when it comes to the analysis of shapes, this is no less important.

In the field of shape and functional data analysis, the computation of a mean on a manifold, is often associated with the Karcher mean. This is generalised from the Riemannian Centre of Mass, first introduced in (Grove and Karcher, 1973). In its simplest form, for a finite set of points, $\{x_1, \cdots, x_N\}$, the Karcher mean, $\bar{\mu}_{\text{KM}}$, can be thought as a solution to the local minimization problem[56], based on some distance, $\mathbf{d}$, in the shape space:

$$\bar{\mu}_{\text{KM}} = \arg_{\mu} \min \sum_{i}^{N} \mathbf{d}(x_i, \mu)^2. \tag{2.53}$$

In this section, we will provide a general overview of the implementation of Karcher means in elastic shape analysis, as well as their usage in enabling further statistical analysis, namely, with principal components analysis.

[55]For example, we ran a test on five pairs of curves, each containing 31 points. On average, the DPA algorithm found the optimal warping function in under a second, whilst the KLR algorithm was approximately 200 times slower.

[56]While the Karcher mean is a *local* minimization, the Fréchet mean can be considered as the *global* minimization. For example, on $\mathbb{R}$, the standard arithmetic mean is analogous to the Fréchet mean, where the distance metric, $d(a, b) = a - b$ for some $a, b \in \mathbb{R}$. The Karcher mean is analogous to the standard point-wise (also called *cross-sectional*) mean, where the distance metric is the standard $\mathbb{L}^2$ metric. This can also be seen in Section 5.1 of the penultimate chapter.

Karcher Means

Recall that our shape space, $\mathcal{S}$, is a quotient space, as defined in Equation (2.34). Thus, as our shapes are equivalence classes, the Karcher mean of square root velocity functions in this space, can also be defined as such:

Definition 2.27 (Karcher Mean) *On a shape space, $\mathcal{S}$, with distance metric, $d_{\mathcal{S}}$, an intrinsic sample mean of n square root velocity functions, $\{q_1, ..., q_n\}$, can be defined as the Karcher mean $[\bar{q}]$:*

$$[\bar{q}] = \arg_{[q]\in\mathcal{S}} \min \sum_{i}^{N} d_{\mathcal{S}}([q_i], [q])^2 \tag{2.54}$$

where, if we recall from Section 2.4.3, $[q]$ can be defined as:

$$[q] := \{((O,\gamma), q) \mid q \in \mathbb{L}^2,\ (\gamma, O) \in \Gamma \times SO(2)\}.$$

Moreover, to better *visualise* the Karcher mean and utilize it in further analysis, we can choose one element $[\bar{q}]$, and call that element the Karcher mean, μ_{KM}.

To illustrate the computation of a Karcher mean, we provide a brief example of an algorithm that can be used to find a Karcher mean of curves, in Algorithm 3. Here, in order to solve the local minimization problem in Definition 2.27, a gradient descent algorithm is employed, as seen in (Dryden and Mardia, 2016), a book covering a concise outline of statistical shape analysis methods. The algorithm begins by selecting an initial mean, $\mu \in \{q_1, ..., q_n\}$. From here, the aim is to align each $q \in \{q_1, ..., q_n\}$ with μ, and to find direction vectors, as once again, as seen in Section 2.5.1, we will be working with the tangent space $T_q\mathcal{S}$, for each q. To compute a distance, d, between some mean μ and a square root velocity function, q, we can incorporate the metric described in Section 2.4.4:

$$(O^*, \gamma^*) = \arg \inf_{\substack{\gamma\in\Gamma \\ O\in SO(2)}} \langle \mu, O(q\circ\gamma)\sqrt{\dot{\gamma}}\rangle_{\mathbb{L}^2}$$

$$q^* = ((O^*, \gamma^*), q)$$

$$d(\mu, q) = \mathbf{d}(\mu, q^*) = \cos^{-1}\langle \mu, q^*\rangle_{\mathbb{L}^2}$$

where O^* and γ^* represent the optimal rotation and reparametrization transformations and $\mathbf{d}$ is the geodesic distance described in Equation (2.28). We can denote $d(\mu, q)$ as θ.

In order to make projections to the tangent space and back, we need to define the exponential and inverse exponential mapping.

Definition 2.28 (Inverse-exponential mapping) *For a geodesic distance, θ, between mean μ, and SRVF q, and a shape space, $\mathcal{S}$, the inverse-exponential map $\exp^{-1} : \mathcal{S} \mapsto T_q\mathcal{S}$ can be defined as:*

$$\exp_q^{-1} q^* = \frac{\theta}{\sin\theta}(q^* - \mu\cos\theta) \tag{2.55}$$

We call $v = \exp_q^{-1} q^*$ the shooting vector or direction vector.

Definition 2.29 (Exponential mapping) *For a direction vector v, mean μ, and a shape space, $\mathcal{S}$, the exponential map $\exp : T_q\mathcal{S} \mapsto \mathcal{S}$ can be defined as:*

$$\exp_q v = \mu\cos||v|| + \sin||v||\frac{v}{||\bar{v}||} \tag{2.56}$$

At each iteration, the algorithm aligns the SRVFs with the mean μ, and computes the direction vectors v. Once this is done for all SRVFs, the average direction vector, $\bar{v}$ is computed[57]. This is the term that we are trying to minimize. Henceforth, a *stopping strategy* is introduced with the term[58] ϵ. If $||\bar{v}|| \leq \epsilon$, then μ will be our Karcher mean. Else, if $||\bar{v}|| > \epsilon$, then μ is updated by projecting $\bar{v}$ from the tangent space to the shape space, using the exponential mapping in Definition 2.29, and going back to the alignment step. This is the basis of the Karcher mean implementation of square root velocity functions, as seen in Algorithm 3. We present some examples of such Karcher mean implementations in Figure 2.13, taken from some of our applications. For more information, we recommend the following literature: (Grove and Karcher, 1973), (Karcher, 1977), the original papers detailing the Karcher mean computations, (Strait et al., 2017) for an outlook of elastic shape analysis of planar curves, including real-world examples of means on mice vertebrae data, and (Dryden and Mardia, 2016), and (Srivastava and Klassen, 2016), for concise textbooks on statistical shape analysis, as well as relevant programming implementations.

[57]Note that since we are now in a vector space, we can compute the standard mean average.

[58]In our applications that involve Karcher mean computations, i.e., in Chapters 4, 5, this constant boundary quantity is set to $\epsilon = 0.0001$.

Algorithm 3: Karcher Mean Example

```
karcher_mean(C,$\epsilon$,$\delta$)

Aim:
Compute the Karcher mean of a set of curves, C, using the dynamic programming algorithm within the SRVF Framework.
Code:
  1. Compute the square root velocity functions of all the curves in C, and compile them into a list, $Qs$.
  2. Pick an initial $\mu$ (this could be the first $q \in Qs$ for example).
  3. Using DPA (Algorithm 1), align the $Qs$ to $\mu$ and find the direction vector $v$. All direction vectors will be added to a list, Vs.
     for $q \in Qs$ do
         (a) $q^*$ = DPA($\mu$,$q$), where $q^*$ is the optimally aligned $q$.
         (b) Compute distance $d = \cos^{-1} \langle \mu, q^* \rangle_{\mathbb{L}^2}$ as in Equation (2.28).
         (c) Get gradient of $d$ by projecting to tangent space and finding the direction vector, $v$.
         (d) Add $v$ to the list, Vs.
     end
  4. Let $\bar{v}$ be the average of the direction vectors in Vs.
     if $||\bar{v}|| \leq \epsilon$ then
         stop
     else
         Update $\mu$ in the direction of $\bar{v}$ and repeat from step 3.
         $\mu \mapsto \cos\delta||\bar{v}||\mu + \sin\delta||\bar{v}||\frac{\bar{v}}{||\bar{v}||}$
     end

return $\mu$ - Karcher mean of the curves, C.
```

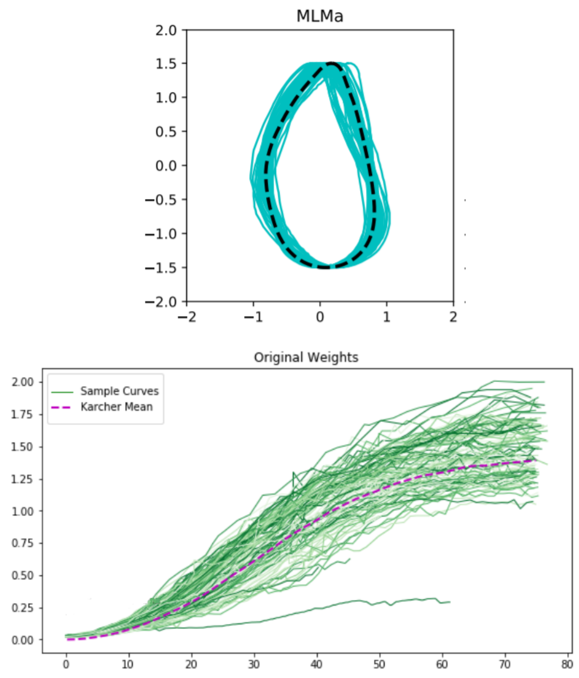


Figure 2.13: Karcher mean examples. Top: Outlines of mussels (cyan) of genus *Modiolus aoteanus*, and a Karcher mean (dotted, black), taken from our Mussel Classification project in Section 4.3. Bottom: Growth curves from kākāpō chicks (green), and a Karcher mean (dotted, magenta); taken from our Kākāpō Health project, which we'll discover in Section 5.1.

Tangent PCA

Another useful tool in statistical analysis is principal component analysis, PCA. PCA is a popular technique to analyse variability within the datasets, used within a variety of topics and concepts, from computing distances between shapes (as will be seen in Chapter 4), to dimension reduction, as seen in (Kambhatla and Leen, 1997).

Intuitively, PCA involves comparing each data point to a mean, to find *principal components*. These principal components are the eigenvectors of a covariance matrix. For example, for n objects, let's consider our data matrix $\mathbf{X}$, where each column $X_i \in \{X_1, ..., X_n\}$ represents one *data-point*. A covariance matrix, $K_{\mathbf{X},\mathbf{X}}$, is found by computing the covariance between every pair in our dataset:

$$K_{X_i,X_j} = E[(X_i - E[X_i])(X_j - E[X_j])] \tag{2.57}$$

where $E[\cdot]$ is the expected value, or *mean*.

From here, K can be decomposed so that we obtain a diagonal matrix Σ containing the principal components for each data-point.

However, we are interested in the shapes of curves, which lie in a rather complicated shape space, $\mathcal{S}$. Therefore, as we require a vector space, we must once again move to working on tangent spaces. Now we can utilize the same tools that we defined to compute Karcher means, in order to perform PCA. As we are working with a tangent space, this leads to the name *Tangent PCA* or *tPCA*.

For a set of n SRVFs, $\{q_1, ..., q_n\}$, the first step in tPCA is to find the Karcher mean, μ_{KM}, for example, using Algorithm 3. Next, for each $q \in \{q_1, ..., q_n\}$, we compute the shooting vector, v, with the definition of the inverse-exponential mapping in Definition 2.28. Henceforth, we define our sample covariance matrix, $\mathcal{K}$, as:

$$\mathcal{K} = \frac{1}{n-1} \sum_{i=1}^{n} v_i v_i^T \tag{2.58}$$

Subsequently, from $\mathcal{K}$, the diagonal matrix, Σ can be found. Henceforth, in order to visualise these *principal components*, we can use the exponential mapping in Definition 2.29, to transform the points back into the shape space.

We present an example of tPCA results in Figure 2.14, taken from one of our projects. As we will soon see in Chapters 4, 5, Tangent PCA and Karcher mean computations can be highly useful tools in analysing shape variability. In the subsequent chapters of this thesis, we will explore these projects and discover more about the practical applications of elastic shape analysis.

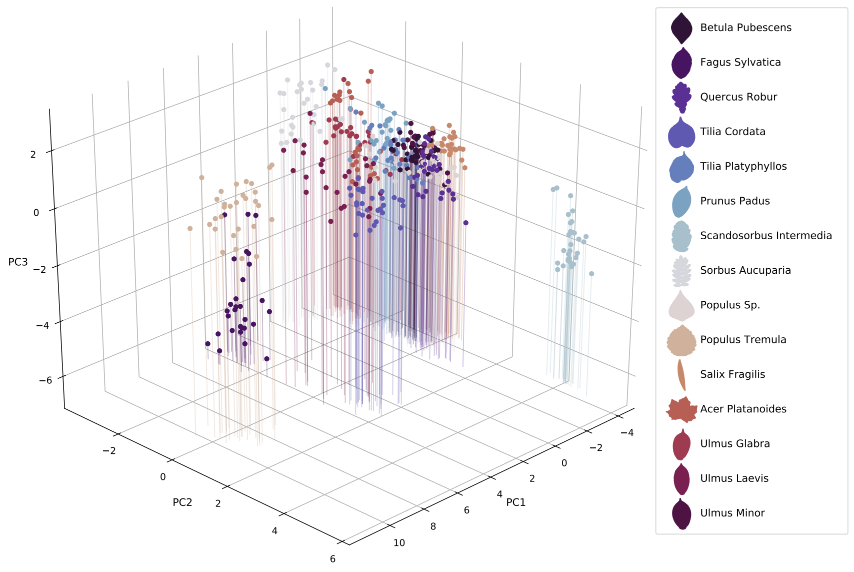


Figure 2.14: Tangent PCA on a set of leaf outlines. Here, we plot the top three principal components for each data point (leaf) in our dataset and colour each point based on its species.

Chapter 3

Image Data Processing

Throughout this thesis, our goal is to enable shape analysis techniques to be used in order to answer questions about the shapes of objects found in varying datasets. In our examples, these questions range broadly from species identification & object classification, to computations of *average* shapes. All of our projects are based on shape analysis applications to two dimensional curve data. The majority of these datasets are comprised of 2D images, of 3D objects. Thus, whilst working on each of our projects, we have developed algorithms to process the relevant image data, in order to extract the necessary curves, from these images. For this reason, we dedicate a chapter of this thesis to a general outlook on tools from image processing that have guided us in extracting curves from image data. Included in this will be a new algorithm that we designed in Section 3.2.3 and a novel framework to extract outlines of objects from images, in Algorithm 6.

3.1 Introduction to Curves in Images

Images of objects can contain a range of information, from the shape of an object and its texture, to its colours, and patterns. Certainly, it can be useful for some datasets to focus on all the information an image holds and thus work on the images as a whole; for example in some medical imaging applications of elastic shape analysis, as seen in (Twining and Marsland, 2003), (Zhang and Fletcher, 2015), (Miller, 2004). However, we recognise objects by their shape (cats are cats whether purely black, tabby, or even hairless) and the texture and colour of the image of an object can vary with the lighting and photographic equipment while the outline remains constant. Therefore our interest lies solely in the *shape* of distinct objects within images, which we assume can be suffi-

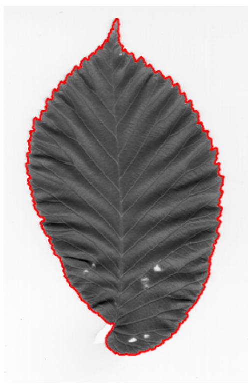


Figure 3.1: An image of an elm leaf from the Swedish Leaf Dataset[59]. Using our own methods, which we will describe in this chapter, we find the outline contour of the leaf and plot it on top of the image (in red).

ciently described by their outlines, rather than requiring the entire image. These are the fundamental reasons why in this thesis we will not be exploring image registration, and instead, we focus solely on the registration of open or closed curves, which may in some applications be extracted from images.

We need to find a way of describing our image data with two-dimensional curves. Assuming that the objects in images have an outline, we look for the curves that represent the outlines of these *main* objects that appear in the images of the datasets. A basic example is shown in Figure 3.1, where we have an image of a leaf (*Ulmus carpinifolia*), and an *outline curve* i.e., a *contour*, plotted in red. The process of finding such an outline is part of the topic of image segmentation. In the general sense, this refers to the *segmenting* of an object from an image. For example, in Figure 3.1, we can think of the leaf as being separated or segmented from the overall image, as an outline contour is found.

3.1.1 Deep Learning for Image Segmentation

The basic motivation behind image segmentation is to partition images to focus on certain aspects within; this plays an important part in further image analyses, such as object detection. Methods of image segmentation can date back to the 1970s, such as in (Brice and Fennema, 1970), which focused on the partitioning of images. In the last decade we have seen an increased focus on neural networks (Priddy and Keller, 2005), to segment

[59]https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/

images. And today, the most common techniques are based on convolutional neural network models, such as (Minaee et al., 2021).

We can think of a more traditional neural network as containing inputs from a series of features that are fed via weighted connections into *hidden* layers, where each node contains some linear combination of the input features, that is then modified by a non-linearity and incorporated in the output. For example, for the leaf in Figure 3.1, features that we could possibly use to classify the leaf could include length and width, or the colour of its veins. One problem that arises here, however, is that we might not know what features are actually useful to us, and so it could be better to use the image itself. The challenge then is that the input is one-dimensional, and any way to reduce the two-dimensional image into one-dimension breaks some of the locality between pixels. This is the motivation behind Convolutional Neural Networks, CNNs, which take multi-dimensional inputs, and use convolution to combine pixels.

Such networks can prove highly useful in image segmentation and in image processing as a whole, hence why are they popular today across various fields. But as with all deep learning methods, they traditionally require a large catalogue of training data; where for image segmentation in particular, we would require not just the original image in the training sample, but a segmented version of it too. Note that the question of *how large* is dependent on a variety of aspects, from the type of data involved and type of deep learning model, to the type of questions needing answered. As an example, the popular *AlexNet* model (Krizhevsky et al., 2017) was trained on around 1.2 million images. Alternatively, approaches of *transfer learning* (Pan and Yang, 2009) are now often taken, which involves the incorporation of a pre-trained model, and allows users to specialise the model for their data using far fewer samples. In this thesis however, as we do not wish to limit the accessibility of our methods by requiring *large* datasets and as some of our datasets are inherently small by nature (for example, the datasets on endangered birds in Chapter 5), we will not be using deep learning methods. For a more detailed overview of deep learning techniques for image processing, we recommend the book (Goodfellow et al., 2016), which provides an introduction to deep learning methods, as well the mathematics behind various models.

3.1.2 Difficulties in Automatic Contour Extraction

Extracting curves from images is not always as simple as it may sound, even with the latest technologies, especially when our aim is to automate the process amongst entire image datasets. While there is image segmentation software that can be used to extract object outlines from images, with just *a few clicks*, such as SHAPE[60,61], they are not sufficiently accurate for automatic extraction. Often, such programs require manual inputs and parameter adjustments for each image, and typically do not output the coordinates of curves into readable formats.

There are a handful of well-known algorithms in computer vision, (some of which we will go into detail about, later on in this chapter) that are commonly used for the task of extracting contours from images, including Geodesic Active Contour (Caselles et al., 1997), and Marching Cubes (Lorensen and Cline, 1987). However, we have found that using one algorithm is usually not enough. Furthermore, each method has its advantages and disadvantages. Though some can be easily used for various languages (for instance, Python and R), and successfully used to find contours (depending on the quality of the images), it is clear that in general, such methods[63] are not designed to be used, on their own, for automatic contour extraction from image datasets. There are various possible reasons for this, such as:

- The default parameters of the algorithm might not suit every image, and will thus require manual tweaking. There is no general way to do this, but parameter selection for certain algorithms has been considered, such as in (Chopina et al., 2013), which discussed parameter optimisation in the Active Contour model, or the in-

[60]`http://lbm.ab.a.u-tokyo.ac.jp/~iwata/shape/`

[61]*SHAPE*, (Iwata and Ukai, 2002), is software based on Elliptic Fourier Descriptors (Kuhl and Giardina, 1982), EFDs, that can be used to extract contours from images. Broadly, for an object in an image, this involves taking the Fourier transform of the boundary points of the object. By adjusting the terms of the inverse Fourier transform, used to go back to the original shape, we can obtain a *smoothed* representation of the boundary curve. *SHAPE* requires two separate programs to process the image and find a contour. Though the software can be fairly quick, it has a disadvantage that it does not output the coordinates of the contours. To do this, a separate program needs to be run in order to find and *save* the normalised elliptic Fourier coefficients. These coefficients can then be used to find the coordinates of the contours, for example, by using functions from the Python library PyEFD[62].

[62]`https://pyefd.readthedocs.io/en/latest/`

[63]To be more precise, we are referring to common contour extraction algorithms found in computer vision, that do not incorporate any deep learning.

troduction of algorithms that attempt to alleviate certain models from necessary parameter estimation, as seen in (Caselles et al., 1995), with the Geodesic Active Contour model.

- Depending on additional attributions of the image (such as size, quality, and colour model), some algorithms that are based on multiple iterations can hold a very long processing time, ranging from seconds to tens of minutes, to process an image.
- Depending on image quality, contours found from contour extraction algorithms may not be smooth. A filtering step is not attributed with all contour extraction algorithms. Though there are certainly exceptions to this, such as the Snakes method (Kass et al., 1988), which contains a specific smoothing parameter.
- Many contour extraction algorithms are designed to find all contours in an image, whereas our aim is to find the single outline contour of the *main* object in an image. As an example, consider the leaf pictured in Figure 3.1. If this image was unaltered, it is highly likely that some contour extraction methods, such as Marching Squares, would not only output the outline contour that we found, but it would also output contours around the white marks within the leaf. Identifying the particular outline contour it then a challenge.

3.1.3 A New Combination of Tools

During this research, we learned that in order to best segment images to find contours, contour extraction methods should be combined with other algorithms, in particular, when the process is to be automated. We were surprised to find that a general *guide* and approach that combines multiple tools is fairly absent from the related literature. This led to the creation of our own combinations of techniques, which we use in various projects, to find outline contours. In almost all data processing used in this thesis, the image segmentation process can be split into three sections:

1. Image binarization
2. Contour extraction
3. Contour smoothing

Establishing a successful procedure to automatically extract contours from images, can play a significant role in the future of image data analysis. We believe that such tools can open doors for those who are less familiar with computer vision techniques, and computer programming in general, by finding a straightforward way of producing coordinates of object outlines, where the single required input is simply the path to a folder of images.

In this chapter, we outline the algorithms used in each of the steps in our image processing procedure. We will briefly go through some well-known algorithms, and we will explore the new algorithms we created, specifically for our projects.

3.2 Image Binarization

3.2.1 What Is an Image?

Simply put, digitised images are represented by pixels that can host various combinations of colours. As an example, we consider the popular RGB colour model, based on the intensities of the primary colours of red, green, and blue. Each of the three primary colours has 256 possible shades (in an 8-bit image), ranging from 0 to 255. Thus, with the RGB model, a two dimensional, coloured image, I_C, can be thought of as an array of pixels, where each pixel $p \in I_C$ is a tuple $p = (i, j, k)$, with $0 \leq i, j, k \leq 255$. Meanwhile, grey-scale images are only concerned with the intensity of the pixels. Therefore, an 8-bit grey-scale image, I_G, can be represented by pixels with intensities $0 \leq p \leq 255$,$\forall p \in I_G$. Alternatively, these can be mapped to $[0, 1]$ trivially, to represent the percentage of intensity, where pixel intensities range from 0 (i.e. black, 0% intensity) to 1 (white, 100% intensity).

3.2.2 Introduction to Image Binarization

Images can hold a myriad of colours. Though this helps preserve images / photographs in their truest forms, it can also lead to the inclusion of noise; this is particularly prevalent in images of poor quality. More often than not, contour extraction algorithms are susceptible to noise, even when the *noise* may not be too obvious to the naked eye, such as subtleties caused by shadows. An example of this can be seen in Figure 3.3, where the contour extraction has mistaken a dark shadow on the lower right side of the vase, for its outline. To try and overcome such issues, we turn to a topic in computer vision: image binarization.

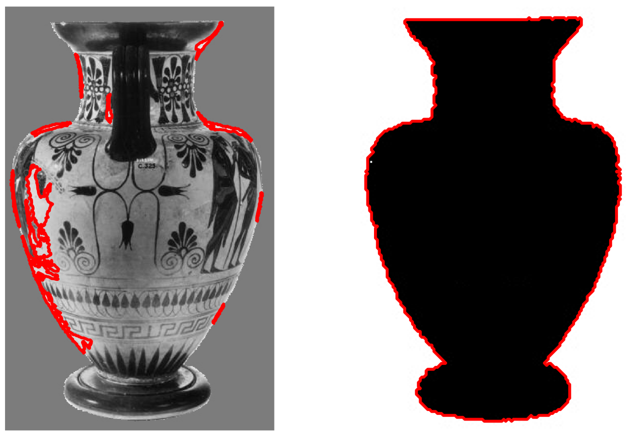


Figure 3.2: Binarization benefits. L: a grey-scaled image, with contours (red) obtained via a Marching Squares algorithm. R: A binarized image, with contours extracted, once again, using Marching Squares. Here, the contour extraction computation on the binarized image was approximately 5.5 times faster than the implementation on the grey-scale image.

Image binarization is the transformation of a multi-tonal image[64] into a bi-tonal image i.e., an image containing precisely two colours, typically, black and white. In computer vision, it is not uncommon to incorporate a binarization algorithm prior to the implementation of a contour extraction algorithm, as it reduces the risk of possible errors in the resulting contours. This is emphasised in the top right plot of Figure 3.3, where a contour extraction algorithm is applied to a non-binarized, grey-scaled image, and it fails to find the contour surrounding the main object in the image. Another example that motivates the need for binarization is seen in Figure 3.2. Not only does the binarized image provide better results from contour extraction, but it can also speed up computations.

Binarization algorithms can be adapted to our liking, for example, by including conditions in order to isolate certain areas of the image, or to emphasise segments. The bottom left plot in Figure 3.3 shows an example of automatic segmentation, where the binarization algorithm has focused solely on the vase and has *wiped out* the other objects in the image. Moreover, the algorithm has *filled-in* the inside of the vase, in order for the contour extraction algorithm to focus on the outline of the vase, and not on the patterns within. This particular example is related to the global aim of image segmentation, which relates

[64]We note that binarization is sometimes exclusively defined on grey-scale images.

to the process of identifying objects[65] in order to partition digitised images into different object and thus find outline contours. This is emphasised in the bottom right plot in Figure 3.3, where the binarization algorithm has not incorporated any segmentation per se, and thus the output also includes contours from other objects, unlike the bottom left plot. Whilst the contours seen in both of these plots, result from contour extraction that was implemented after image binarization, we notice that it has not escaped the issues of noise, particularly the noise caused by the shadow seen to the right of the vase. This tell us that image binarization is not always perfect. However, had an image binarization algorithm not been implemented at all (as seen in the top right plot of the same figure) the results would have been far worse.

3.2.3 Image Binarization Algorithms

In order to binarize images, most algorithms take a thresholding approach. Broadly, *thresholding* refers to the transformation of pixels into white or black, based on some *optimum* threshold(s). For example, consider the mapping $H : [0,1] \mapsto \{0,1\}$, with an optimal threshold, ρ^*, applied to a grey-scale image I, with pixel intensities, p, ranging from 0 to 1:

$$H(p) = \begin{cases} 0, & \text{if } p \leq \rho^* \\ 1, & \text{otherwise} \end{cases} \tag{3.1}$$

Image thresholding methods can often be split into three categories: global thresholding, local thresholding, or a combination of the two. Global thresholding methods search for one optimal threshold to transform the pixels in an image, as in Equation (3.1). Meanwhile, local thresholding methods transform pixels using a threshold that is based on the local neighbourhood to each pixel, and not on the entire image. Furthermore, there also exists hybrid methods which combine the two.

Otsu's Method

Today, there are many image binarization algorithms from the local-thresholding method introduced in (Niblack, 1985), to the hybrid binarization method in (Kuo et al., 2010). Out of all binarization algorithms in computer vision, one algorithm is without a doubt

[65] In a computer-vision sense, we can think of *objects* in images as clusters of pixels in an image that are notably different from the surrounding pixels in the background.

[66] https://www.wgtn.ac.nz/slc/about/our-programmes/classics/classics-museum

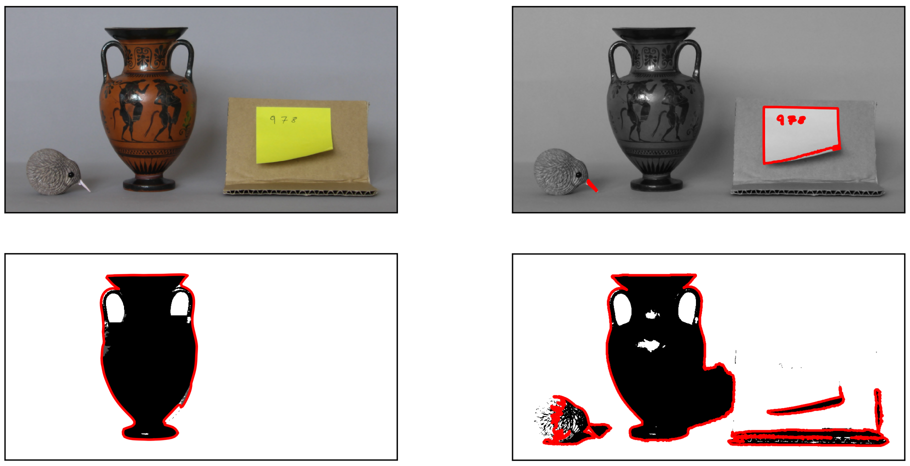


Figure 3.3: Top left: a photograph of an ancient Greek vase (alongside a kiwi toy for scale and some card to label the picture), taken by Stephen Marsland, in the Classics Museum[66]at Victoria University of Wellington. Bottom left: a binarized version of the original photograph, and an outline contour of the vase, overlaid in red. Our own binarization algorithm, which also incorporates the essence of segmentation, is used in this particular example, which we will outline in the subsequent sections. Top right: a grey-scale version of the original photograph, and the top contours obtained after contour extraction on the grey-scale image. Bottom right: contour extracted and plotted on top a binarized version of the original photograph, using a popular binarization algorithm, Otsu's thresholding method, (Otsu, 1979). Although our binarization algorithm has done rather well in binarizing the original image, the contour extraction algorithm was still susceptible to some noise caused by a shadow, which can be seen on the lower-right side of the vase. And the same can be said for Otsu's thresholding method, where the resulting vase contour is more affected by the shadow. However, the success of both binarization algorithms, particularly our one, can certainly be praised, when the resulting contours are compared to the contours in the top right plot, where a method of image binarization was not employed prior to contour extraction. The basis of the contour extraction steps on all images was the Marching Squares algorithm, (Lorensen and Cline, 1987), which we will describe in more detail, later on in this chapter.

the most notable: Otsu's method, (Otsu, 1979).

Otsu's method, also known as Otsu's thresholding method, is based on global thresholding. The algorithm involves splitting the pixels in an image into two classes, using an arbitrary threshold value. From here, the aim is to optimize the threshold value by minimizing the *within-class* variance, whilst maximizing the *between-class* variance. More specifically, for a grey-scale image, I, Otsu's method creates two histograms based on the frequencies of pixel intensities in each of the two classes, I_1, I_2, split by some threshold, t. These histograms are then used to compute the variance σ_i^2 for each class, and hence the within-class variance, V_{wc}, and the between class variance, V_{bc}, where $i \in [1, 2]$:

$$V_{wc} = \frac{n_1}{N}\sigma_1^2 + \frac{n_2}{N}\sigma_2^2 \tag{3.2}$$

$$V_{bc} = \sigma - V_{wc} = \frac{n_1 n_2}{N^2}(\mu_1 - \mu_2)^2 \tag{3.3}$$

where n_1, n_2 are the number of pixels within each class, $N = n_1 + n_2$ is the total number of pixels, σ is the total variance, and μ_1, μ_2 are the means computed from the histograms. Furthermore, the derivation of the final equality in Equation (3.3) can be found in (Otsu, 1979). Next, the algorithm goes through all possible values for t (i.e. based on the histogram bins, from the starting bin, to that of maximum intensity) until it finds the optimal t^*:

$$t^* = \arg\max_t V_{bc} \tag{3.4}$$

Finally, the pixels are transformed using the optimal threshold, t^*, just as we saw in Equation (3.1), and hence the image is binarized.

Algorithm 4: Otsu's Method

`otsu_binarize(I,min_intensity=0,max_intensity=255)`

Aim:

Binarize a grey-scale image, I, with Ostu's thresholding method.

Code:

```
1. Set top_t = 0 and top_$V_{bc}$ = 0.

2. Perform lobal thresholding on image, $I$.
   for $t \in$ min_intensity, $\cdots$, max_intensity do
       Split pixels into classes $I_1, I_2$ and create histograms:
       for $p \in I$ do
           if $p \leq t$ then
               $p \in I_1$
           else
               $p \in I_2$
           end
       end
       Compute pixel means $\mu_1, \mu_2$ from each of two histograms,
       and probabilities, $\rho_1, \rho_2$, such that $\rho_i = \frac{n_i}{N}$ for each $i \in$
       $\{1, 2\}$, where $N$ is the total number of pixels, and $n_1, n_2$
       are the number of pixels in $I_1, I_2$, respectively. Hence,
       compute the between class variance, $V_{bc} = \rho_1\rho_2(\mu_1 - \mu_2)^2$.
       if $V_{bc} \geq$ top_$V_{bc}$ then
           top_t = $t$
           top_$V_{bc}$ = $V_{bc}$
       end
   end

3. Let $I_B = I$. Transform each pixel $p \in I_B$ based on threshold, top_t.
   if $p \leq$ top_t then
       $p$ = min_intensity
   else
       $p$ = max_intensity
   end

return $I_B$ - Binarized version of the image, $I$.
```

Novel Binarization Technique

Image binarization can be applied to a whole variety of images, for varying purposes, from forms and documents, used in digital scanning apps, to medical imaging. Though there exists a plethora of techniques, it's often that they each seem better on certain types of images, and perform less successfully on others. For example, see the comparisons made between Otsu's Method and others in (Munshi and Mitra, 2012) on fingerprint binarization and the comparisons of hybrid and local thresholding methods on the binarization of leaflets with complex backgrounds in (Kuo et al., 2010).

For the most part, the data that we deal with in the various projects of this thesis share one commonality; namely, that the datasets consist of images, where the outline of the prominent object (positioned approximately in the centre) is our sole interest. This implies that we are not interested in the other objects in the image, or in the image as a whole - indeed, we are only focused on that *main* object, and in particular, on its outline. For this reason, it is in our interest to employ a binarization technique that not only *segments* that prominent object, but also covers the inside of the main object, in order to enable us to focus on the object's outline. Therefore, we create our own binarization algorithm that takes on board such matters, with the aim of applying it to the datasets that we encounter in the following chapters of this thesis.

The algorithm outlined in Algorithm 5 describes our image binarization method. Fundamentally, the algorithm is based on three parts: thresholding, segmentation, and *object-filling* (in other words, *painting* the entirety of the inside of the object in one colour). The main difference between our method and other algorithms is the inclusion of object-boundary ellipses in order to binarize the images. Two ellipses are computed. The larger ellipse is used for the segmentation step, by recolouring all the pixels outside of it white. This, in turn, separates out the object, from the rest of the image. Meanwhile, the smaller ellipse is used in the *object-filling* step, by finding the left and right-most black pixels on a *quasi*-thresholded image, that are within the smaller ellipse. Subsequently, all pixels with those bounds are turned black.

Algorithm 5: New Binarization Technique

```
new_binarize(I,white_or_black='black',black_bound=0.2,
 white_bound=0.15,prop=0.45,border=20,ellipse_leeway=15,
 small_diam_leeway=0.05,large_diam_leeway=0.45,center=1)
```

Aim:

Binarize a grey-scaled image, I, and cover the inside of the main object.

Parameters:

- I - grey-scale image.
- `white_or_black` - Options: {'black', 'white', 'both'}. This parameter is used in the final transformation of pixels, where any pixel that has not, thus far, been transformed into white or black, is transformed into `white_or_black`. If `white_or_black` is `'both'` then two binarized images are returned.
- `black_bound (float)` - A float ($0 \leq$ `black_bound` ≤ 1) to compute a proportion of the maximum intensity that is used to create upper and lower bounds, for the initial criterion of turning pixels black.
- `white_bound (float)` - `white_bound` is similar to `black_bound` except it is used in the initial criterion of turning pixels white.
- `prop (float)` - Upper bound for proportion of black pixels in I.
- `border (float)` - Percentage of the image size, to create a border around the image, where all points in the border, are turned white. In other words, `border` determines the subsequent *thickness*.
- `ellipse_leeway (float)` - When the vertex and co-vertex of ellipses are computed, the variable `ellipse_leeway` is used to add some *leeway* to the positions of these vertices. `ellipse_leeway` is the percentage of the image size that is used for the leeway.
- `small_diam_leeway (float)` - Additional leeway, based on a proportion (determined by `small_diam_leeway`) of the image size, added to the diameter of the small ellipse.

- `large_diam_leeway (float)` - Equivalent to `small_diam_leeway` but used for the leeway to the diameter of the larger ellipse.
- `centre` - If `centre` is `1` then we assume that the object of interest is centred within the image. Otherwise, set to the rough position of the object's center, within the image, `center = [x,y]`.

Code:

1. Find the average colour of the background, of the image, I.
2. Compute the approximate position of a vertex and co-vertex of an ellipse, by finding the position of the top, bottom, left, and right-most *dark* pixels, starting from the assumed centre of the object. By doing this, we can gain a rough idea of the boundary of the object, which we can use to compute the ellipses.
3. Check whether the height of the assumed boundary, H_B, is almost the height of I, H. If the height is found to be *large*, we will require a thinner border, thus the variable `border` must be increased, by a percentage.

```
if $\frac{H_B}{H} > 0.75$ then
    if $\frac{H_B}{H} > 0.95$ then
        border = border+20
    else
        border = border+10
    end
end
```

4. **Thresholding Step I:** Create two boundaries (`bound_w`, `bound_b`) to be used to convert pixels into white or black, based on the average background colour and the variables `white_bound`, `black_bound`. Subsequently, all pixels with a value within `bound_w` will be turned white, whilst all pixels with values outside of `bound_b` will be turned black. In other words, we assume that pixels whose colour differs the most from the background colour form part of the object, hence why pixels outside of the boundary are turned black. Moreover, all pixels whose positions fall within the border (created with the variable `border`), will be turned white. Label the new image as I_1.

5. If the proportion of black pixels, ρ, is high, then the first thresholding step is repeated, with different variables:

```
if $\rho >$ prop then
    if $\rho > 0.95$ then
        If 95% of the image is covered in black pixels, it is
        likely that something has gone wrong. This is often
        caused by the colour of the main object being lighter
        than the background colour, whilst white_bound <
        black_bound. In this case, we swap the two variables.
        black_bound,white_bound =
        sort(black_bound,white_bound)
    else
        black_bound = black_bound+0.1
    end
    Redo Thresholding Step I and update $\rho$.
end
if $\rho > 0.95$ then
    Redo Thresholding Step I with black_bound doubled.
end
```

6. Create a small ellipse, E_S, and a large ellipse, E_L, based on the respective diameter leeway variables, and on the vertex / co-vertex values, as we computed in the second step.

7. **Thresholding Step II:** Turn all pixels in I_1 whose position falls outside of E_L white. Call this new image, I_2. By doing this, we can *segment* the main object of interest, and *ignore* the background. Next, we aim to *fill-in* the inside of the object, by iterating through the y-positions that appear in the smaller ellipse, Y_{E_S}:

```
for $i \in Y_{E_S}$ do
    positions_b = np.where($I_2$[i,:]==0)[0]
    if len(positions_b) > 1 then
        $X$ = $x$-positions in positions_b that are within $E_S$.
        if len($X$)>1 then
            $I_2$[i,$X$[start]:$X$[end]] = 0
        end
    end
end
```

8. **Thresholding Step III:** Find all pixels in I_2 that are neither black nor white, $I_{\bar{B}\cap\bar{W}}$. Compute the average intensity of the pixel in $I_{\bar{B}\cap\bar{W}}$ and the standard deviation, in order to create a new lower bound. Hence, all pixels in $I_{\bar{B}\cap\bar{W}}$ whose intensity is less than the lower bound, will be turned white. This new image is called I_3.

9. **Thresholding Step IV:** Once again, find all remaining points that are neither black nor white, $I_{\bar{B}\cap\bar{W}}$, this time in I_3. Let $I_4 = I_3$, and iterate through the positions of the pixels in $I_{\bar{B}\cap\bar{W}}$.

```
for $(i, j) \in I_{\bar{B}\cap\bar{W}}$ do
    neighbourhood = $I_3$[i-10:i+10,j-10:j+10].flatten()
    $\alpha$ = len(np.where(neighbourhood=mx)[0])
    where mx is the maximum intensity (usually 0 or 255).
    if $\alpha >$ len(neighbourhood) $\times 0.5$ then
        $I_4$[i-2:i+2,j-2:j+2]=mx
    end
end
```

10. **Thresholding Step V:** Create a binarized image, I_B, by recolouring all remaining pixels in I_4, that are still neither black nor white, into black or white (depending on the variable `black_or_white`).

`return` I_B - Binarized version of the image, I.

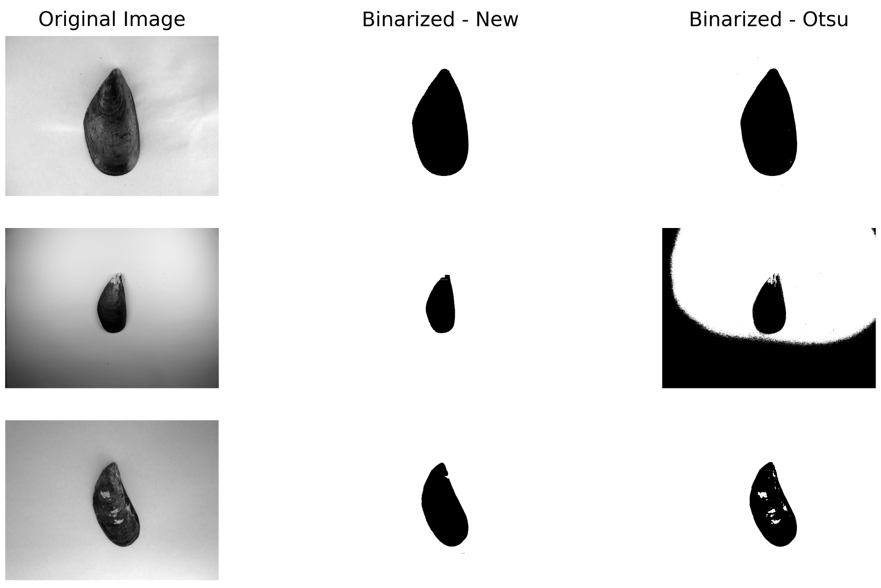


Figure 3.4: Binarization examples. Here, we test out Ostu's Thresholding Method, and our new binarization technique, in order to binarize images of mussels. Overall, both methods have done fairly well in binarizing these simple images, particularly the first image. Though the second image may seem similar to the others, it consists of darker shadowings that have clearly affected the success of Otsu's method, whilst our algorithm seems fairly unaffected. Meanwhile, in the last image, there is a *mistake*, albeit rather minuscule, from our binarization method, as it has missed part of the top right side of the mussel. Otsu's method does not make that same mistake. As Otsu's method is not designed to *fill-in* the objects of an image, it is not surprising that is hasn't done so in this last image, as the light patterns on the mussel are certainly well-camouflaged with the background of the image.

3.3 Contour Extraction

Recall that our interest is in the outlines of objects presented in images. These outlines, also known as *contours* in Computer Vision, describe the boundary of objects, using coordinates (which can be interpreted as (x, y)-coordinates) that can refer to the positions of the pixels in the image where that boundary is found. There are numerous contour extraction algorithms, many of which are commonly used in medical imaging applications, such as the Active Contour Model, used to detect boundaries in echocardiograms, (i.e. heart ultrasounds), in (Chalana et al., 1996).

3.3.1 Marching Squares

Of all contour extraction algorithms, one algorithm in particular, features quite a lot in literature, due to its simplicity, speed, and efficiency, namely, the Marching Square algorithm, (Lorensen and Cline, 1987).

Throughout this PhD, we have extracted contours from various types of images, from digitised versions of noisy photographs taken from the early twentieth century, to spectrograms computed from audio files. Although these images greatly differ from one another, we find that the Marching Squares Algorithm constantly excels in extracting contours from our datasets. It has therefore become a staple tool in our image processing steps.

The Marching Squares algorithm for two-dimensional contour extraction, is based on its three-dimensional equivalent, the Marching Cubes algorithm, first described in (Lorensen and Cline, 1987), to form three-dimensional visualisations of objects from medical images. Whilst the Marching Cubes method incorporates a grid of cubes in its process to find contours, the Marching Squares algorithm, analogously, assigns a grid of squares.

In order to find contours from images in our datasets, we use the Marching Squares algorithm, implemented by the Python library, scikit-image[67], (van der Walt et al., 2014). For more details of this algorithm, we recommend (Hansen and Johnson, 2011) or (Maple, 2003). In general, this algorithm can be split into 4 steps but as we input binarized images in our projects, in place of grey-scale images, the steps can be reduced to 2. Nonetheless, we briefly outline the four original steps here:

[67]`https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html`

Step I – Binarization The Marching Square method is commonly used on grey-scale images. The first step of the algorithm is to binarize the image using some threshold value. If the input is already a binarized image (as it is in our case), the choice of the threshold is redundant.

Step II – Grid Formation A grid of squares is overlaid on the binarized image, where each square (also called *cell*) covers a 2×2 pixel block, with the cell vertices each centred within a pixel. In other words, the vertices of each cell, are, in total, contained within four pixels. An index is then created for the vertex nodes, depending on the binarized state of the pixel that they are inhabited within. For example, if a vertex node is within a black pixel, the node is coloured black, and so on.

Step III – Finding Contours From here, the aim is to check whether parts of a contour (or contours), pass through each cell, and if so, *how*. To answer this question, a look-up table, such as the table shown in Figure 3.5, is incorporated, in order to determine what the contour lines would look like in each case. There are $2^4 = 16$ possible cases for what a cell looks like, based on the colour of its four vertex nodes. For example, in Case 0, as there are no black pixels (and hence black nodes), we assume that a contour does not pass through this cell. Similarly, in Case 15, as the cell is completely within a block of black pixels, we assume that the cell is not positioned on the boundary of an object in an image, and thus, a contour does not pass through. We iterate through each node in the grid, to determine whether a contour passes through the cell comprising of the given node in its top right corner, and *draw* the relevant lines, based on the look-up table.

Step IV – Linear Interpolation At this stage, the contour lines exclusively go through the midpoints of the squared grid. If the input image is not originally binarized, then the start and end points of contour lines through the cells can be adjusted. This adjustment is done using a linear interpolation on the original values of pixels. More specifically, we consider a vector that a contour line touches, and find the original pixel values of the two nodes that form that vector. A value, μ, is then computed, based on the original pixel values and the threshold value used in the first step. Henceforth, the vector is linearly interpreted at μ, to find the new start or end position of the contour line. We reiterate that this step can only make a difference when the original image is not a binarized image.

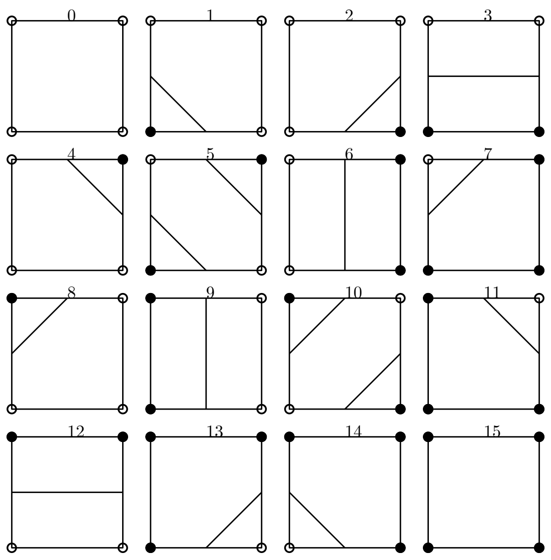


Figure 3.5: Contour lines look-up table, for use in the Marching Squares algorithm. Each square, or *cell*, comprises of a 2×2 pixel block, where the vertices are centred within four pixels. The colour of the vertex nodes are determined by the colour of the binarized pixel it inhabits, either white (shown here with a *clear* circle), or black. Thus, there are 16 possible cases for what cells could look like, labelled here from 0 to 15. This look-up table is used to establish how a contour line may pass through a cell, in order to find contours. Contour lines pass through cells via the midpoints of the vectors (grid-lines), and they only *touch* the vectors that are connected by one white node and one black node. With this look-up table, we can establish piecewise linear contours from images.

The Marching Squares algorithm is an efficient method for finding piecewise linear contours from images. As with all algorithms, it does have its flaws, such as its lack of attention to *finer* details and its inability to produce contours containing *sharp* corners, a problem addressed in (Gong and Newman, 2013). Nonetheless, it still remains a very popular algorithm for contour extraction, particularly in applications of topography or medical imaging, see, for example, it's use in analysing CT scans in (Huang et al., 2011). Next, we will take a brief look at another popular method of contour extraction, and see how it compares to the Marching Squares algorithm.

3.3.2 Snakes

Contour extraction algorithms can vary greatly from one another, from methods that incorporate meshes and pre-built lookup tables, as in the Marching Cubes algorithm, to methods that are based on more mathematical tools such as splines, as seen in *Snakes*.

Snakes, also known as the Active Contour model, were first described in (Kass et al., 1988). With its ability to work well with *noisy* images, and its inclusion of a smoothing constraint, Snakes has become one of the most popular methods in Computer Vision, particularly for its practical use in segmentation, edge detection, and contour extraction.

Minimizing the Energy

The primary focus of the Snakes model is to fit a spline, or *snake*, to the boundary of an object in an image, by minimizing an energy defined by the spline and by the actual image. Intuitively, we can think of the snake as an elastic band which is initially stretched and extended around the desired object in an image, that is then released to tightly *wrap* the object, thus defining the object's boundary. This *release* can take numerous iterations, until the shape of the *elastic band* (the snake) no longer changes (or until a maximum iteration number is reached). Hence, there are two important implications. Firstly, the algorithm requires a *rough guide* of where the contour may lie. And secondly, the algorithm will always output <u>one</u> contour, unlike the Marching Squares algorithm, where there is no limit (see for example, the right-side plots of Figure 3.3). Though we briefly outline the method, for more information on how Snakes works, we refer the reader to (Ivins and Porrill, 1995), as well as the original paper, (Kass et al., 1988).

As described in (Kass et al., 1988), we can represent a snake with the parametric equation, ν, and thus describe it's energy functional as follows:

$$\nu(s) = (x(s), y(s)) \tag{3.5}$$

$$\begin{aligned} E^* &= \int_0^1 E_{\text{snake}}(\nu(s))ds \\ &= \int_0^1 E_{\text{int}}(\nu(s)) + E_{\text{image}}(\nu(s)) + E_{\text{ext}}(\nu(s))ds \end{aligned} \tag{3.6}$$

where E_{int} is the snake's internal bending energy, E_{image} is the energy defined by the image, and finally, E_{ext} is the energy of the external forces, which are specified, traditionally, by the user.

For the algorithm to act like an elastic band, two constraints are introduced, in order to define the snake's internal bending energy.

$$E_{\text{int}} = \frac{1}{2}\left(\alpha(s)\left|\frac{\partial \nu}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2 \nu}{\partial s^2}\right|^2\right) \tag{3.7}$$

for some chosen weights, $\alpha(s), \beta(s)$. Note that both parts of E_{int} focus on the change of position of the snake, at every iteration. The first part can be thought of as the tension or elasticity, making sure that all points in the snake move at the same time. Meanwhile, the second part is focused on the *stiffness* of the snake, including its *smoothness*, by taking care of possible *spikes*. If we are interested in a spike or corner, $\beta(s)$ can be set to zero to accommodate this.

The energy of the image, E_{image}, can be interpreted as the weighted sum of forces used to *navigate* the snake towards certain features, namely, lines, segments, and terminations (of line segments and / or corners):

$$E_{\text{image}} = \omega_{line} E_{\text{line}} + \omega_{edge} E_{\text{edge}} + \omega_{term} E_{\text{term}} \tag{3.8}$$

with chosen weights[68] $\omega_{line}, \omega_{edge}, \omega_{term}$. For example, if we wish for our snake to be attracted towards brightness, positive values are used for ω_{line}, whilst, alternatively, negative values are used to attract the snake towards dark lines. In the simplest form, the three energy functionals seen in Equation (3.8) can be defined as:

$$E_{line} = I(x, y) \tag{3.9}$$

$$E_{\text{edge}} = |\nabla I(x, y)|^2 \tag{3.10}$$

$$E_{\text{term}} = \frac{\partial \theta}{\partial \mathbf{n}_\perp} \tag{3.11}$$

where I represents the intensity of the image, $\theta = \arctan \frac{C_x}{C_y}$ is the gradient angle, $\mathbf{n}_\perp = (-\sin\theta, \cos\theta)$ defines the unit vectors perpendicular to the direction of the gradient, and C is the curvature of level lines:

$$C(x, y) = G_\sigma(x, y) \cdot I(x, y) \tag{3.12}$$

[68]Note that the weights in E_{image} are referred to by variables of the same name, in scikit-image's snake function, `active_contour`[69].

[69]`https://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.active_contour`

for the Gaussian G_σ, with standard deviation σ.

Finally, the energy of the external forces, E_{ext}, depends on what *our* aim is for the algorithm. For example, if the snake is initialised outside[70] of the desired object, we want it to be attracted inwards, in order to eventually wrap the outline of the object, as we described earlier in this section. Thus, an external force can be used to control this *attraction*:

$$E_{\text{ext}}(\nu(s)) = \kappa|(x, y) - \nu|^2 \tag{3.13}$$

for a point (x, y) on the image, and the *spring*[71] factor variable, κ.

In order to minimize the energy defined in Equation (3.6), the Snakes algorithm employs an iterative gradient-descent approach. On each iteration, the snake moves a few steps closer to its *optimal* position, whilst also being smoothed. Here, both the smoothing constraint and the maximum number of *steps* taken in each iteration, is decided by the user. Once the snake is aligned with the desired object boundary in the image, or when the maximum iteration is reached, the algorithm stops and outputs the final version of ν.

Comparisons & Uses

In comparison to the Marching Squares algorithm, the Snakes algorithm is much more complex and requires a lot more input from the user. Its success and processing time are highly dependent on two things in particular:

1. The original snake, initialised at the start.

2. The maximum number of iterations.

Moreover, these are correlated with each other. As an example, if the initialised snake is not anywhere near the shape of the boundary (for instance, if it were simply a circle surrounding the main object) it may not reach the desired boundary, before the maximum number of iterations is reached. Nevertheless, the results produced by a Snakes algorithm, particularly when time is not of the essence (and thus a large maximum number of iterations can be chosen), are often successful. This is seen in Figure 3.6, where we

[70]It is also possible for the initial snake to placed within the boundary of the desired object. In this case, the snake *repels* and moves outwards, until it reaches the boundary.

[71]κ is the spring constraint for a spring connecting a point on the image to a point on the snake.

compare a contour found using Snakes, with contours found using a Marching Squares algorithm on the same image.

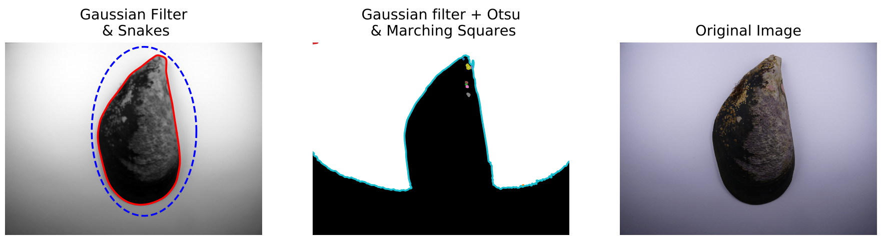


Figure 3.6: Snakes vs Marching Squares – 1. Left: As a Snakes algorithm is commonly applied to filtered images, we use a Gaussian filter to smooth the image slightly before applying Snakes. Moreover, we set the initial snake as an ellipse, shown here in blue. Finally, we plot the resulting contour in red. Centre: In order to make it a *fair* comparison, we also use the Gaussian-filtered image for the Marching Squares test. Here, as per the required thresholding step for non-binarized images in Marching Squares, the optimal threshold value is found using Otsu's algorithm, which is hence followed by Marching Squares contour extraction. Right: Original image of mussel. We note that the *downfall* of the Marching Squares algorithm in this particular example is due to the dependency on the thresholding step, which has been heavily affected by the noise caused by the shadowing in this image. The Snakes algorithm, on the other hand, is scarcely affected by the shadows, and thus performs well in finding the mussel's contour. This emphasises that Snakes can work well on noisy images.

The most difficult aspect of Snakes is undoubtedly the selection of its parameters. Though there have been suggestions on how to address this issue, as in (Rousselle et al., 2003), where a genetic algorithm is used to optimize the parameters, it is very often the case that the parameter selection remains a manual and *fiddly* step. When used for the purpose of contour extraction, any slight change in the parameters of a Snakes algorithm, can have a big effect on the resulting contour. Furthermore, as manually tweaking the parameters can take a long time, Snakes is generally not the most convenient method to incorporate in automatic contour extraction.

Though the use of Snakes to extract contours may seem arduous, it could in fact be incorporated with a different purpose. In particular, we can employ a Snakes algorithm to *smooth* our contours. Recall that the algorithm includes a smoothing of a snake at each iteration, based on a smoothing constraint. If a contour has already been estab-

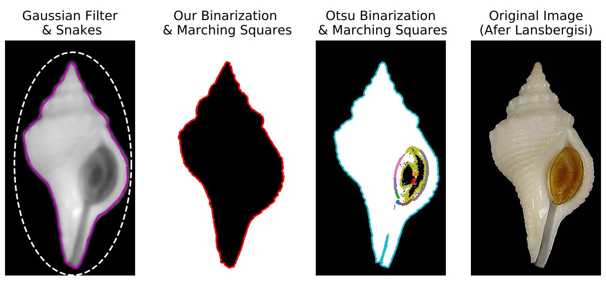

Gaussian Filter & Snakes
Our Binarization & Marching Squares
Otsu Binarization & Marching Squares
Original Image (Afer Lansbergisi)

Figure 3.7: Snakes vs Marching Squares – 2. In this example, we take a look at a naturally-more-complicated shape. This time, we not only compare the different contour extraction methods, but we also compare the two binarization methods we previously discussed. Far left: A Gaussian filtered image, with an ellipse as the initial snake (white), and the final contour (magenta). Centre-left: Image binarized using our binarization method, as described in Algorithm 5. Contour (red) extracted using the Marching Squares algorithm on the binarized image. Centre-right: Image binarized using Otsu's algorithm, and contours, once again, extracted using Marching Squares. Far right: original shell image. Here, the results show that both the Marching Squares method and Snakes perform well in extracting the contour around the shell. On closer inspection however, it seems that the Snakes method has missed some tight corners in the top right part of the shell. This is likely due to the snake over-smoothing. Meanwhile, both binarization techniques have also worked fairly well, with our binarization algorithm coming out on top, as unlike Otsu's method, it did not fail at the bottom tip of the shell.

lished, then we can set the initial snake to be this very contour. Henceforth, by iterating the Snakes algorithm only a handful of times and with the algorithm's smoothing constraint set very low, we can smooth the initial contour, without the risk of over-fitting, and without enabling the additional parameters chosen for the Snakes algorithm to have a significant effect on the shape of our contour. The *stopping strategy* is a pre-chosen number of iterations; since this is very low, the model does not overfit; the aim is only to smooth the contour. An example of this can be seen in Figures 3.8 and 3.9, where the incorporation of Snakes has improved the original contour. We note that a similar approach was also taken in (Yang and Marchant, 1996), where Snakes was used to *refine* contours surrounding fruit blemishes.

As the maximum number of iterations is set to a small number[72], this method of smooth-

[72]For example, in some projects we set the maximum to be $2-5$. This is far smaller than the default number used in scikit-image[73]'s function, which is set at 2500.

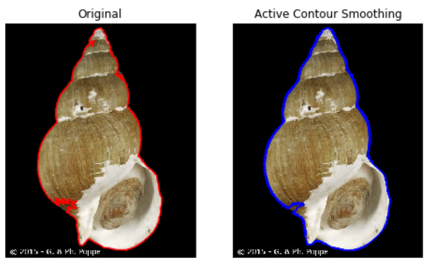


Figure 3.8: Snakes for smoothing. Left: An example of a contour obtained via a Marching Squares algorithm (red). Right: A smoothed contour (blue) using the Snakes algorithm, with a few iterations. Here, the contour on the left was used as the initial snake. We can see that though the smoothed contour is not perfect, it has certainly improved the original contour.

ing will be computationally quick, and will prove an efficient addition to the automatic image processing we take on for our projects, specifically, when smoothing closed curves. Though this method may seem more complicated than traditional smoothing techniques, such as splines, in general, Snakes not only performs better on more complicated shapes, but also it does not impose restrictions on the initial contour, unlike certain spline methods. For example, many implementations of splines in the Python library `scikit-image`, require the original curves to be strictly increasing in the time domain[74]. However, such problems do not exists with a Snakes method. Furthermore, if the original contour had missed the object boundary in certain areas, a Snakes method may be able to overcome this as it has the additional information of the image in its implementation, unlike spline methods which only require a contour in their implementation.

3.4 Experiments

In this chapter, we have visited four algorithms, each of which will play some part in the shape analysis projects that we will be describing in the remaining chapters. An algorithm, or in most cases, a *combination* of these algorithms, will be used to find curves from our datasets, that will subsequently be incorporated with elastic shape analysis, in one way or another, in order to answer questions regarding the shape of the objects.

[74]Thus, contours will need to be split into sections before these methods can be used for smoothing.

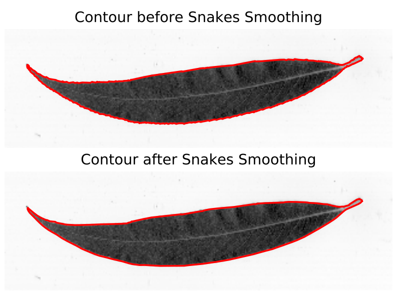

Contour before Snakes Smoothing
Contour after Snakes Smoothing

Figure 3.9: Smooth contour extraction with Algorithm 6, with a processing time of around 2.7 seconds[75], from image loading to contour smoothing.

3.4.1 Our Contour Extraction Approach

The datasets and the *type* of datasets we work with are all different, as are the questions we ask of them. Thus, the way in which we will be using these algorithms will also differ. The data processing step for each dataset will be discussed in the relevant project chapters / sections. However, in Algorithm 6, we provide an outline of what the simplest image processing step looks like.

Here, the algorithm is designed for basic automatic contour extraction, given the location paths of images in a dataset. It follows a 5-step process to obtain a smooth contour describing the outline of the desired object for each image. More specifically, we find the contour using Marching Squares on a binarized image, which is then *smoothed* using Snakes. Furthermore, in case multiple contours are found during the Marching Squares step, in this example an assumption is made that the longest contour (in terms of the number of points), represents the boundary of the object. This is because our image binarization algorithm aims to segment the object whilst masking the inside of the object, thus the longest contour should in theory be the boundary we seek.

3.4.2 Comparisons to Other Methods

In order to further evaluate whether our approach to contour extraction would work better than other methods when it comes to the automatic extraction of object outlines from 2D images, we designed an experiment based on an assortment of images.

[75]On a machine with an Intel Core i7-8550U @ 1.80GHz CPU and 8GB of RAM.

Algorithm 6: Automated Image Processing Example

```
find_contours(paths,binAlg=new_binarize,$N$ = 3)

Aim:
Find all contours from a dataset of images.
Initial Step:
Create a list containing the location paths of the images in the dataset, paths.
Code:

  1. Set all_contours = [].

  2. One by one, find all contours.
     for $path \in paths$ do
         Step I: Load grey-scaled version of image.
         Step II: Binarize image with chosen binarization algorithm.
         img_bin = binAlg(img)
         Step III: Find outline contour.
         C = marching_squares(img_bin)
         $j = \arg\max_i \text{length}(C_i)$
         outline_contour = C[$j$]
         Step IV: Smooth contour.
         smoothed_contour =
         Snakes(img,initial_snake=outline_contour,max_iterations=$N$)
         Step V: Add contour to list of other contours.
         all_contours.append(smoothed_contour)
     end

return all_contours - Return a list of smoothed outline contours.
```

This experiment was split into two sections. The first part focused on an array of popular methods as well as our own method, and on a random selection of object images, ranging from images of *buccinidae* to amphorae, as seen in Table 3.1. Whilst the second experiment was based on an image dataset of just one object, and on a finer selection of methods.

Our first experiment was based on the following methods:

1. **Method 1: Otsu & Marching Squares** – Otsu's thresholding technique (Algo-

Object Type	# Samples
Amphorae	21
Lekythoi	5
Mussel Shells	21
Other Gastropods	3

Table 3.1: Total number of sample per object, in Experiment 1.

rithm 4, (Otsu, 1979)) was employed to binarize the image, and this was followed by the Marching Squares algorithm (Section 3.3.1, (Lorensen and Cline, 1987)) to find contours.

2. **Method 2: Canny Edge Detection & Marching Squares** – Here, we employed a Canny edge detection algorithm[76] (Canny, 1986) which produces a binarized image with edges highlighted. This was then combined by a Marching Square algorithm to extract the contours.

3. **Method 3: Snakes** – We used a Snakes algorithm (Section 3.3.2, (Kass et al., 1988)) to find the outline contour of the object. Recall that this method requires an initial snake as its input. Thus, as we are working with a random assortment of images, we chose an arbitrary initial snake. These were ellipses centred in tfhe centre of the images, with the diameters based on proportions[77] of the image size. Finally, in order to save time in our experiments, the maximum number of iterations for this method was set at 500.

4. **Method 4: Canny Edge Detection & Border Following** – Here, we used the Canny Edge Detection algorithm to create a binarized version of the image. Subsequently, a Border Following algorithm (Suzuki et al., 1985) was employed to find contours, using the popular OpenCV[78] implementation.

5. **Method 5: Our Method** – Lastly, we used our own approach of extracting outline contours, as seen Algorithm 6.

[76]https://scikit-image.org/docs/stable/auto_examples/edges/plot_canny.html
[77]Specifically, this was 70% for the x-diameter, and 90% for the y-diameter, as we were mainly working with landscape images.
[78]https://docs.opencv.org/3.4/df/d0d/tutorial_find_contours.html

Method	% Success
1	44%
2	8%
3	8%
4	28%
5	60%

Table 3.2: Experiment 1: Percentage of *good* outline contours found per method. Note that an outline contour is classified as *good* if the experimenter judges that it correctly resembles the boundary of the main object in the image.

In order to compare the different approaches, we implemented the 5 methods on a dataset of 50 images, and plotted the contours overlaid onto the original image. Subsequently, we analysed the plots produced by each of the image, by eye, and counted those that had correctly identified the outline of an object from a contour. Note that for the methods that find multiple contours (i.e., Methods 1,2,4), when making our judgements, we ignored the contours that were not outline contours. Thus, we defined the *success* of each method by its proportion of correctly-identified contours, i.e., contours that correctly[79] outlined the main object in the image.

The results of our experiment can be seen in Table 3.2. Some methods certainly performed better than others and this could be down to various reasons, from the default parameters not being suited our particular dataset, or due to cap on the maximum number of iterations. We also reiterate that the images in our dataset of purposely of varying quality, from high-quality images of gastropod shells, to images of mussels that contained a lot of shadows. We display some plots from this experiment in Figure 3.10. Furthermore, as we find that Methods 1,4,5 performed the best in these tests, in our second experiment we focus on these 3 methods.

In our second experiment we concentrate on one class of objects, namely, images of mussel shells. Furthemore, as we are focusing one class, we can tweak the parameters based on these objects. For example, for Method 5, we increase the `border` parameter in the

[79]For example, in Figure 3.10, Method 3 did <u>not</u> result in a *correct* outline of the amphora in the image ,whilst Method 5 <u>did</u> produce a contour that correctly outlined the amphora.

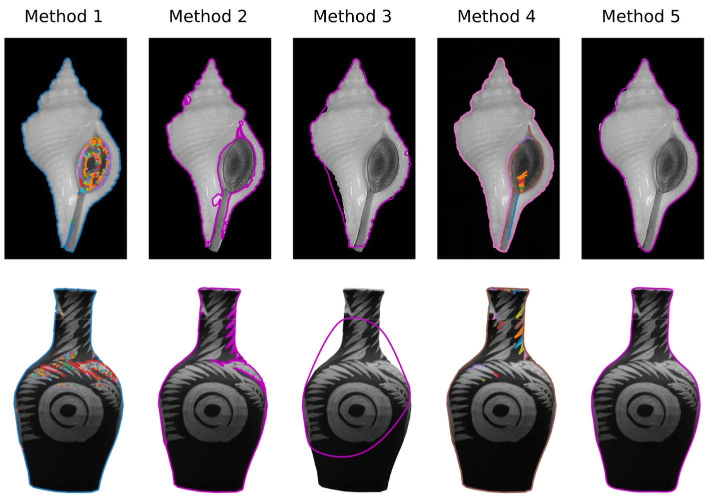


Figure 3.10: Experiment 1: Example contours, from all five methods. We display images of buccinidae shell and a modern-day amphora vase, with contours overlaid. Here, we see that almost all the methods performed well in extracting an outline contour from the shell image. However, the contours resulting from Methods 1 and 4, moved within the shell itself. Furthermore, the contour from Method 5 may be slightly *over-smoothed*. The amphora however, seems to be complicated by the distinct patterns within the vase, which has affected almost all of the methods, with the exception of Method 5, which incorporates a *boundary-filling* step, as discussed in Section 3.2.3.

binarization step (Algorithm 5), as our dataset consisted of a small mussel shell centred in a rather large image, as seen in Figure 3.11. We conduct our tests using the same technique as the first experiment. The results for this experiment (shown in Table 3.3) reveal that our approach (Method 5), outperformed the other methods, and correctly extracted the outline contour in 80% of the mussel images. These results further motivated us to employ our contour extraction techniques for our specific projects.

Next, we delve into our projects, where we will demonstrate some results of the image processing methods we have discussed in this chapter. And we will show that with the reasonable combination of computer vision, elastic shape analysis, and machine learning, we can discover some exciting results.

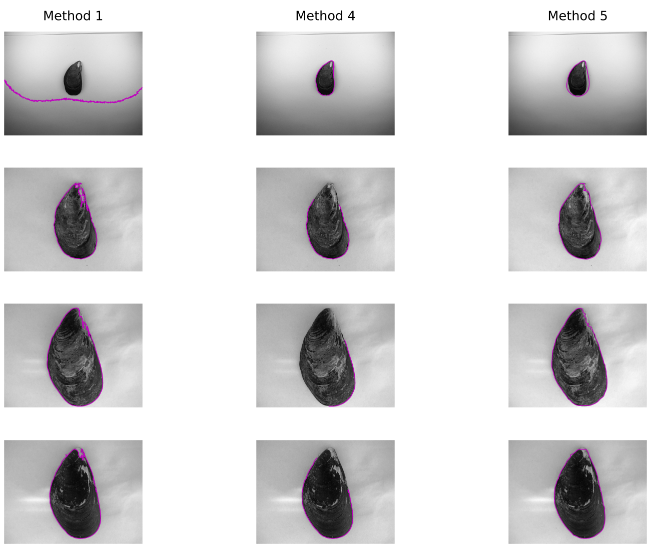


Figure 3.11: Experiment 2: Contours overlaid onto images of mussel images. As Methods 1 and 4, can find multiple contuors on an image, here we plot only the *longest* contour, as we assume that the longest contour represents the outline contour. In these specific plots, Method 5, resulted in most accurate contours, despite some of them not being perfect (such as the first mussel contour). Method 1 also performed rather well, except the slight distortions based on the patters within the mussel. Meanwhile Method 4, in these specific plots, failed to find a closed outline contour, and instead found open curves representing a segment of the outline.

Method	% Success
1	48%
4	12%
5	80%

Table 3.3: Experiment 2: Percentage of *good* outline contours found in the top methods. Here we see that our contour extraction approach, based on Algorithm 6, performed the best.

Chapter 4

Applications of Elastic Shape Analysis to Closed Curves Extracted from Images

By analysing image datasets with mathematical methods we are able to extract information about image collections for a wide variety of practical uses. In this chapter, to analyse image datasets, we incorporate elastic shape metrics and we define the *shape* of an object from an image, by the closed curve describing its outline. We want to show that elastic shape metrics are useful in quasi-automatic applications on image datasets. Our research is broadly based on three aims:

- Classification of objects using their outlines, with comparisons between various methods, and human experts.
- Comparison of image-derived outlines and measurements from the true object.
- Demonstration that shape alone may not be enough.

Henceforth, we take on three collaborative projects, to explore the practical applications of image data analysis, by analysing shapes with elastic shape analysis. In the order of which they will appear in this thesis, the projects are the following:

1. **Classification of Ancient Greek Vases** – We explore comparisons between Elastic Shape Analysis methods with traditional methods of Geometric Morphometrics by primarily analysing their effectiveness in classification. We focus on Greek vases, but our datasets entail:

(i) Ancient Greek vases,

(ii) Gastropod shells,

(iii) Swedish leaves.

This is a large collaboration, with collaborators spread across the world, including biologists, archaeologists, and malacologists.

2. **The Effectiveness of Images** – In this project, we study the effects of employing images to analyse the shapes of vases, in comparison to the traditional method of hand-measurements, and the more modern technique of 3D laser scanning.

3. **Classification of Mussels** – Our last project is focused on using elastic shape analysis to distinguish non-native mussels from native mussels found in New Zealand.

4.1 Classifying Ancient Greek Vases

The classification of historical and archaeological objects has played a key role in shaping our understanding of history. As a simple example, the classification of objects found on an archaeological site can help guide archaeologists in dating the site. Whilst historically classification was done by eye, as seen in some examples in (Beazley, 1956) to identify specific painters of vases, in modern literature, techniques have ranged from using geometric morphometrics to study ancient Brazilian spear heads in (Okumura and Araujo, 2019), to using a neural network model to classify ancient Persian cuineform characters, as seen in (Mostofi and Khashman, 2014). Our main interest, however, is on one specific set of historical objects: the vases of Ancient Greece.

4.1.1 The Shapes of Vases

From ancient Persia to Babylonia, and ancient Greece to the Byzantine empire, many ancient civilisations relied on pots or vases, for various reasons, such as practical use, ceremonial use, or ornamental use. Thus, given their wide-spread usage, and their durability, it is no surprise that countless numbers of vases have been discovered today, with many originating from ancient Greece. By classifying such vases, we can begin to understand about their usage and origins and hence learn more about the ancient civilisation they originated from.

It is estimated that around $100,000$ painted ancient Greek vases survive today. These come in a variety of styles and shapes, with some vases being more bizarre than others, from rhytons[80] shaped as animal heads, to the elegant wine jug, epichysis, as can be seen in Figure 4.2. These varying styles were transmitted between potters for generations, resulting in the vast array of Ancient Greek vases we see today. Therefore, there is a lot of interest in the classification of vases, particularly to discover who made the vases, how they have been constructed, what their usage is, and which period they belong to. To answer such questions today, archaeologists and historians often work with images of vases, as a whole.

The decorations on vases are very often relied on when it comes to the classification of

[80]Rhytons were conical vessels mainly made for drinking, and were particularly popular in Ancient Persia, although the Ancient Greeks are also known to have made them.

these ancient vessels. Additional attributions that can also play a part in vase classification are the materials used to construct the vase, and the handles. The outline of the vase however, is seldom used in classification. Though there are some examples of literature that focus on vase shapes, such as (Bloesch, 1940), the outline of an object alone (i.e., without any additional information) is scarcely utilised. Given the occasional complexities of factors that can affect the outline shape of the vase (for example, intricate handles), combined with the similarities in shape across various other classes of vases, it is probably not too surprising that the outline shape has not been used as a primary feature in the classification of vases. Take Figure 4.1 for example, showing a neck amphora (left) and a volute krater (right). If the handles are removed, then except for the slightly elongated leg of the volute krater, the two vases may start to look rather alike. This challenge motivated us to ask the question: *If the shapes of vases are indeed difficult to classify by eye, can an algorithm succeed instead?*

Figure 4.1: An ancient Greek neck amphora (L) and a volute krater (R) with Beazley IDs 4529 and 1006971 respectively. Though the outlines of the two vases are clearly different, if the handles were to be removed, their outline shapes would be rather similar.

4.1.2 Shape Analysis for Vase Classification

Ancient Greek vases come in all shapes and sizes. The peculiarities of vase shapes, the wide range of data, and academic curiosity all inspire our research into vase classification using solely the shape of the vase. This research focuses on three primary questions:

1. Is shape alone sufficient for the classification of ancient Greek vases?

2. Which shape analysis methods have the best accuracy?

3. Are the results using shape alone comparable to human experts?

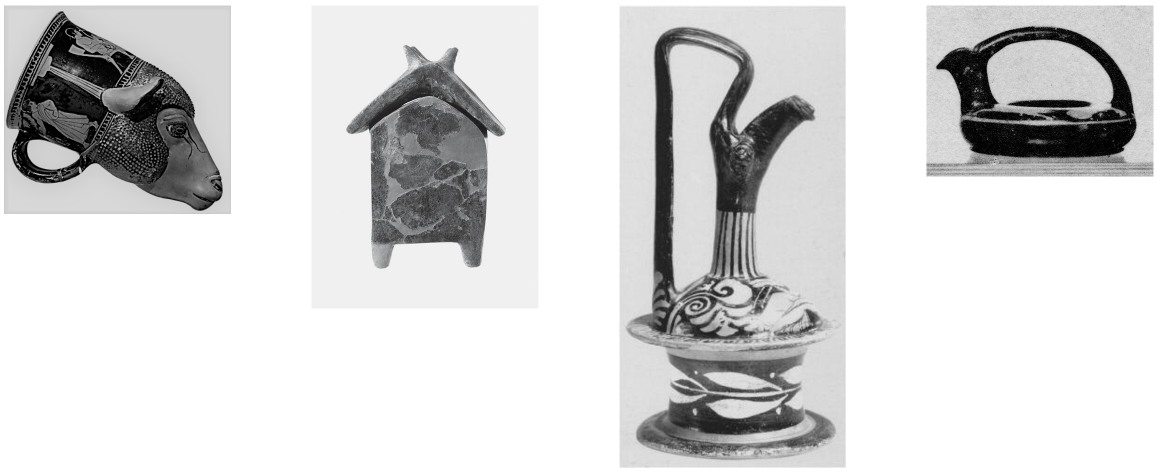


Figure 4.2: Assortment of vase shapes. L-R: Lamb rhyton, urn, epichysis, askos. We note that these particular types of vases were not included in our classification tests. This was primarily due to the lack of any type of shape consistency in these classes. For example, urns are named for their purpose, and can hence take a variety of different shapes, thus, it is not intuitive to consider its shape in classification tests.

In collaboration with Prof. Armand Leroi of Imperial College, we began a project based on the classification of ancient Greek vases using solely the outline shapes of vases, extracted from a collection of vase images. In the subsequent sections of this chapter, we outline our approach in more detail, from describing the process of finding vase outlines, to detailing the various classification algorithms.

4.1.3 Data

Our focus is on the outlines of vases from images. Henceforth, in this project, we utilize images of individual ancient Greek vases from the Beazley Archive[81]. The archive holds hundreds of thousands of vase images from museums and personal collections from across the globe, as emphasised in Moffett (1992), a paper on the archive's contribution to global accessibility of humanities data. We selected a sample of vase images from a restricted list of different classes (though a larger-scale project, on a wider variety of classes, is planned). Our collaborator vetted the images in the sample to make sure of their suitability, for example, to check if images contained a whole, complete vase, presented in *front-view*[83].

[81]The Beazley Archive[82]was built on the archive of the famous archaeology professor, Sir John Beazley, known for his work on the classification of Ancient Greek vases.

[82]https://www.beazley.ox.ac.uk/carc/pottery

[83]The archive contains all types of vases images, including broken vases and fragments.

The true classes (or *types*) of the vases were independently verified.

Vase Outlines

The extraction of vase outline contours from the images incorporates Algorithm 6, described in Chapter 3, which involves binarizing an image and extracting a contour from the binarized image, which is then smoothed. However, for this project, the process is not so simple, as two additional steps are also required. Firstly, as we only interested in the *main shape* of the vase, we must either create a contour extraction algorithm that *ignores* the handles, or we must remove the handles. Secondly, as the classes of vases we study are generally symmetrical, we want our vase contours to also be symmetrical along the vertical axis. By doing so, we provide additional smoothing, whilst also removing structures of the vase that we do not care about, such as vase spouts. In the next sections, we explore the methods used to find smooth, symmetric outlines from images of ancient Greek vases, in the following order:

1. Handle removal algorithms.

2. Binarization.

3. Contour extraction, smoothing, and symmetrization.

4.1.4 Removing Vase Handles

Elaborate lids and impressive handles often appear in ancient Greek pottery. One might argue that these are the features that characterize many vases, such as the majestic volute krater seen in Figure 4.1. When a potter creates a vase, the body of the vase is morphed from clay along with the basic mouth and base. As handles and other features are added externally afterwards, for our shape classification, it is the primary body of the vase that we are fundamentally interested in.

It is vital to remove the *external* features of a vase during the data processing stage of our project. Some vases are photographed from an angle that excludes such features, and others might not have any external features in the first place. But there will always be a few vases in any sample of images that include handles and lids and so on. As our sample was quite small (circa 700 individual vase images), the simplest way to achieve this would be to manually edit the image of the vase (using a photo editing tool) before starting the image thresholding and contour extraction process. Though this is the avenue we occasionally resorted to, it was preceded by a series of algorithms we developed to automate the process of handle-removal. Here, we will briefly outline the basis of these algorithms and discuss the potential they could have in the future.

Excluding a small minority of classes, in general ancient Greek vases with handles can be split into two categories: those with handles that start and loop back onto the body such as an amphora, and those with handles that stick out, such as bell-kraters and some cups. The former group is, unsurprisingly, much more complex than the latter, as initial contours need to be found that not only describe the outline of the vase, but also the contours inside the handles (as displayed in green and red, in the left plot of Figure 4.3). Moreover, this *handle contour* can be segmented even further, as one side of it is part of the vase outline (which we label as the *inner handle contour*, plotted in green in Figure 4.3), whilst the other side is part of the external handle (plotted in red in Figure 4.3). Therefore, we create two primary handle-removal algorithms that extract an outline contour, with the handles excluded, from the two discussed *groups*.

Algorithm 7 is a sketch of a contour extraction function that extracts a contour from an image of a vase without the handles. This function can be used on vases with a handle that loops back onto a vase, such as a regular amphora or an askos vase (see Figure 4.2). An example of how this algorithm works can be seen in Figure 4.3. On the left we see the original outer contour, the outer handle contour (red – runs along the external handle), and the inner handle contour (green – runs along the vase outline). The image on the right shows the new contour which excludes the handle, as well as the positions on the contour, where the handle is attached to the vase (labelled as `(a,b)` in Algorithm 7, and plotted in magenta and blue respectively, in Figure 4.3).

An outline for the handle-removal of Group II vases, i.e., vases with handles that stick out, can be seen in Algorithm 8. This function is simpler than Algorithm 7, as its primary aim is to find the start and end positions of the handle on the original outline contour. Subsequently, a straight *cut* is made along the outline contour between the assumed start and end points. An example of this can be seen in Figure 4.4, where the outline contour

[84] https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html
[85] https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html

Algorithm 7: Handle Removal – Group I

```
handle_removal_1(bin_image, one_side=True)
```

Aim:

Given a binarised image, extract an outline contour of a vase from its image, excluding its handles. This is used for *Group I* vases, i.e. those with looped, amphora-like handles.

Code:

1. Find all contours in the image, for example, by using scikit-image's[84]contour extraction function, `all_contours = measure.find_contours(bin_image)`.
2. Search through `all_contours` to find the outline contour, `(x,y)`, (the longest contour that goes around the handles), and the handle contours on each side. These contours are based on approximations of the locations of the handles, and the range / length of the individual contours.
3. Choose one side, and segment the *inner handle contour* from the original handle contour.
4. Find two positions on the outline contour corresponding to where the external handle is attached to the body of the vase. We call these two positions `(a,b)`.
5. Adjust the inner handle contour, based on the coordinates `(x[a],y[a])` and `(x[b],y[b])`, and call this new contour segment `(xh,yh)`.
6. Create a new outline contour, `(X,Y)`, by going along the original outline contour in the areas that don't include the handle, but going through the updated inner handle contour in the areas which originally aligned with the handle. For example, for one side, `X,Y = (x[:a]+xh+x[b:],y[:a]+yh+y[b:])`.
7. Repeat steps on other side, if necessary:

```
if one_side==False then
    Repeat steps 3-6 for the other side of the vase.
    Let (X,Y) represent the outline contour which
    connects both sides of the vase.
end
```

`return X` - x-coordinates of new outline contour.

`return Y` - y-coordinates of new outline contour.

Algorithm 8: Handle Removal – Group II

`handle_removal_2(bin_image, one_side=True)`

Aim:

Given a binarised image, extract an outline contour of a vase from its image, excluding its handles. This is used for *Group II* vases, i.e., those with handles that *stick out*.

Code:

1. Find all contours in the image, for example, by using scikit-image's[85]contour extraction function, `all_contours = measure.find_contours(bin_image)`.
2. Find the longest contour in `all_contours`, `(x,y)`.
3. Choose one side to focus on. This can be the *best* side (determined by certain contour factors), or a pre-selected side.
4. Find the start point, `a`, of the handle on the outline contour. To do this, we identify *indents* on the outline curve. These are defined as points on the original outline curve where the x-coordinate is greater than both its preceding and succeeding neighbouring points.
5. The end point, `b`, of the handle is found by creating a boundary for the handle position (using the start point found), and searching for points with similar x-coordinates as the handle start point.
6. Create a new outline contour that excludes the handle, for example, on one side: `X,Y = (x[:a]+x[b:],y[:a]+y[b:])`.
7. Repeat steps on other side, if necessary:

```
if one_side==False then
    Repeat steps 4-6 for the other side of the vase.
    Let (X,Y) represent the outline contour which
    connects both sides of the vase.
end
```

`return X` - x-coordinates of new outline contour.

`return Y` - y-coordinates of new outline contour.

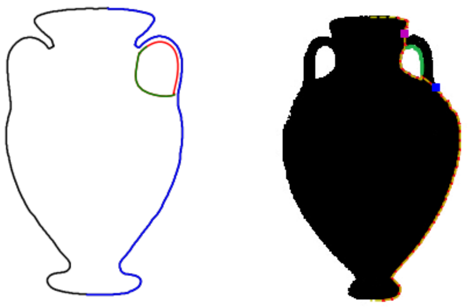


Figure 4.3: L: A contour around an amphora vase. Focusing on one side of the vase (blue), a handle contour is found which is subsequently segmented as the outer handle contour (red) and the inner-handle contour (green). R: A different (*panathenic*) amphora, with the handle contour on one side (green), and the outline contour, excluding any handle (yellow, dotted red). Also plotted, are the two positions (pink and blue) on the outline contour, (i.e. `(a,b)` in Algorithm 7), which are the assumed positions where the external handle attaches to the vase body.

(excluding the handle) can be seen in red, whilst the start and end positions of the handle are plotted in blue.

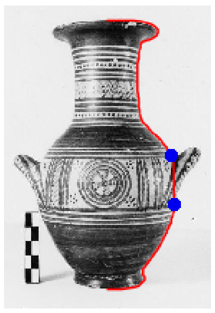


Figure 4.4: Outline contour example using Algorithm 8. Here, we see the output of the algorithm as a contour on one side of the vase, with the handle excluded, imposed onto the original vase image. We also plot the assumed start/end points of the handle (blue) that the algorithm uses in order to *cut* the handle from the original outline contour.

We outline the two principal algorithms that find contours that exclude the vase handles. However, many sub-algorithms were also made, in order to improve results for certain classes, for example to focus on askos vases (see Figure 4.2) due to their *looped* handles appearing on the top of their vase bodies, as opposed to the *sides* as in amphorae.

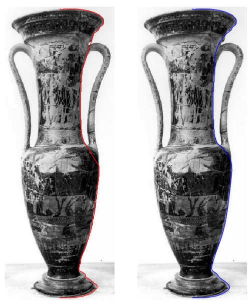


Figure 4.5: A loutrophoros vase that has had its handle removed using Algorithm 7, where its output of is superimposed onto the original vase image. L: straight cuts have been made at the two positions where the handle is joined to the vase body. R: splines have been used instead; where we can see a noticeable difference, particularly in the centre, when taking a closer look.

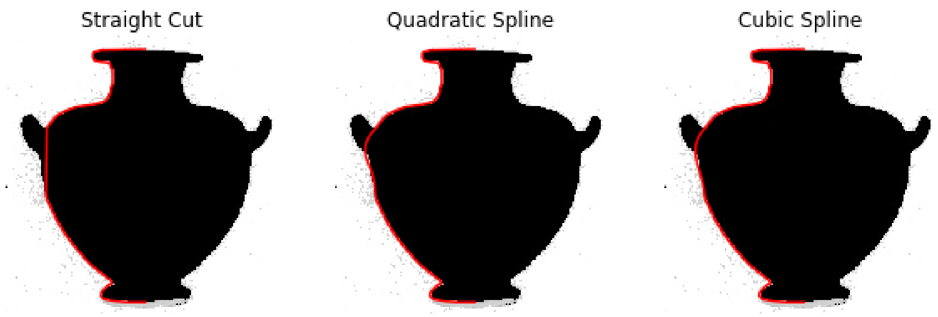


Figure 4.6: Smoothing outline contours of Group II vases. In this example we have a binarized Hydria vase, with an outline contour (excluding the handle) on one side, found using Algorithm 8, and plotted in red on the image (far left). On the left we have the original output, which performed a regular straight cut between the start and end positions of the handle. In the centre figure we fitted a quadratic spline instead, whilst on the right we fitted a cubic spline.

In order to test the performance of all of our *handle-removal* algorithms, we worked on a large sample of vase images (circa 2000) of differing quality, with a multitude of classes, separate from the sample used for our specific classification project. Overall, our success rate was around 55%, where this score was based on a manual classification of contours, by eye. In other words, contours that did not correctly outline the boundary of a vase were deemed *bad* (see Figure 4.8 as an example), whilst those that correctly outlined the vase were classed as *good*.

Definition 4.1 (Good / Bad Contours) *A contour that correctly represents the boundary of an object within an image is defined as good, else the contour is classed as bad.*

Though this score is not too bad, there are many possible reasons for the errors. In many cases, the algorithms failed because of poor image quality, though other reasons

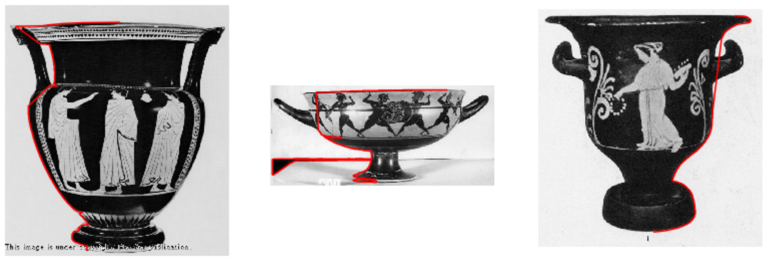


Figure 4.7: Bad vase contours. We show examples of where the handle removal methods of Algorithms 7 and 8 failed to find an appropriate outline contour. L: The handle removal algorithm has done a decent job in finding an outline contour that excludes the handle, however, it has mistakenly included a small portion of the handle at the top of the vase. This is a rather understandable mistake to make, given that the handle is only really distinguishable from the vase lid by its colour. Additionally, the outline contour has also included some text that appeared at the bottom of the image. Centre: In this example, the original contour included a dark spot that appeared on the vase stand. This affected the resulting outline contour that excluded the handle. R: Here, the only mistake of the handle removal algorithm is its failure to properly locate the start and end positions of the vase handles on the outline contour.

were down to failures of pinpointing the exact start and end positions of handles, or to additional objects appearing in the image, and so on. An example of such *bad* contours can be seen in Figure 4.7. Furthermore, we note that in some cases, though the resulting contour was not too bad, the straight cuts made by either Algorithm 7 and 8 were simply not particularly credible. One possible solution to this specific problem is to use splines instead of straight cuts. An example of these using Algorithms 7 and 8 can be seen in Figure 4.5 and Figure 4.6 respectively.

We note that by taking a different approach, for example, employing a neural network model, we may be able to improve the performance and obtain better vase outlines that exclude handles. However, such techniques would certainly require much larger datasets, considering the variety of outline shapes seen across ancient Greek vases. On the other hand, we have no doubt that by continuing to work on the algorithms that we have already created, we can one day achieve more impressive results. But given the performance of the handle removal algorithms in our tests at the time, we made a pragmatic decision to supplement our results by using image manipulation software to correct the output.

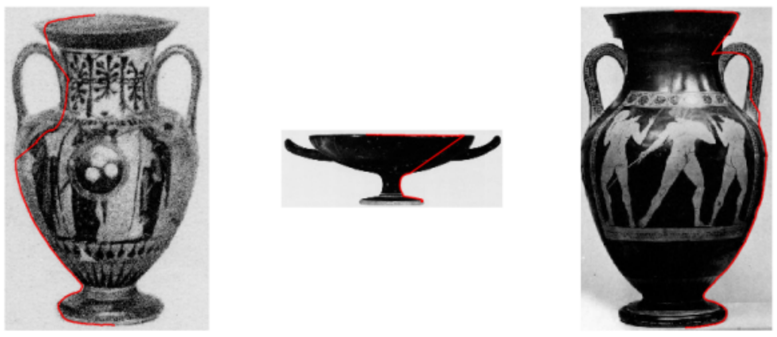


Figure 4.8: *Unsuccessful contours.* The contours in this example, as seen in red, overlaid on the vases, were all deemed *bad* in our tests of the handle-removal algorithms.

4.1.5 Vase Image Binarization

Binarization is a form of image thresholding. It involves a modification to an image such that each pixel is one of two colours: in our case, black or white. Thus, the outcome of binarization of an image is a black-and-white version of the original image. As discussed in Chapter 3, binarizing an image is a worthwhile pre-processing step before contour extraction. Not only does binarization accelerate the process and progress of contour extraction algorithms (for example, by denoising an image by default), but it can also be used to mask certain regions of an image. The latter can be utilized to *fill-in* the inside of a vase, so that the decorations within the vase are not detected by the subsequent contour extraction algorithms.

In section 3.2.3 of the Image Processing chapter of this thesis, we discussed a novel binarization technique we designed in order to binarize images containing one object centred in the middle. Here, we use a section of this algorithm, aimed at segmenting the vase, and masking the inside of the vase in one colour. This refers to part of Algorithm 5, up to *Thresholding Step III*; therefore, here, we call this *sub*-algorithm the Binarize-Fill Method. During this project, to binarize our sample of vase images, we employed this Binarize-Fill Method, and often combined it with a well-known thresholding method, Otsu's Method, (Otsu, 1979), in order to obtain the best binarization results. Though we discussed both of these algorithms in great detail in Algorithm 5 and 4 in Chapter 3, we briefly outline the two methods here:

Binarize-Fill Method: The function starts by detecting the average *background* colour. It uses this average to create two initial thresholds, t_1 and t_2, used for mapping pixel intensities; $P_{\leq t_1} \mapsto 0$, whilst $P_{\geq t_2} \mapsto 1$, where P represents the set of pixels in the image, 0 is used for black, and 1 is white. Based on the new set of pixels, two ellipses are created, E_1 and E_2 where $E_2 \subset E_1$. All pixels whose positions in the image fall outside of E_1 are subsequently turned white (1). Meanwhile, the interior of the object is masked by finding the furthest dark pixels from the centre to the boundary of E_2, and thus turning all the pixels within those positions black (0). The pixels whose positions lie in between E_1 and E_2 go through a nearest-neighbour-like procedure, where their new pixel colour is based on their neighbouring pixels. For further details on this method, we refer the reader to Algorithm 5.

Otsu's Method: Otsu's method involves finding a threshold, t, that is used to split the pixels into two classes. Here, the aim is to minimize the *within-class* variance, V_{wc}, and maximize the *between-class* variance, V_{bc} (as described in equations (3.2) and (3.3), respectively). The algorithm goes through all possible values for t (based on the colour model of the image) until it finds the optimum, $t^* = \arg\max_t V_{bc}$. Finally, the pixel intensities are updated based on the optimal threshold, for example, $P_{<t^*} \mapsto 0$, whilst $P_{\geq t^*} \mapsto 1$, for the set of pixels P. For more details, see Algorithm 4 of Chapter 3.

4.1.6 Finding Vase Outlines

Recall that our aim is to construct a dataset of smooth and symmetrical vase outlines from images of ancient Greek vases. Where our classification project differs[86] from many other projects, specifically in relation to data processing, is that we aim to establish a process that automates the procedure of obtaining contours from images.

Contour Extraction

As mentioned previously, the images in our sample were vetted and modified using image-editing software to manually remove features such as handles, lids, and stands, when necessary. Following this, images were binarized with the methods discussed earlier. The next step was to extract the outline contours from the binarized images. We used scikit-image[87]'s implementation of the Marching Squares algorithm (see Section 3.3.1) to

[86]For example, there have been many cases of classification projects on shapes of objects from images, where their forms are obtained by *hand*, or, alternatively, algorithmically obtained for each image individually.

[87]`https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html`

extract all possible contours. Moreover, as this algorithm outputs all contours that it can find, we filter for the outline contour by choosing the contour with the longest range (in the vertical axis).

Smoothing & Symmetrization

Once an outline of the vase has been found, our next step is to obtain a *symmetrical* outline. The vases chosen for our dataset should all exhibit bilateral symmetry, vertically down the centre. Though this is simple to see when looking at the images, for the resulting outlines from the contour extraction step, we first have to make the assumption that the outline contours do indeed correctly *outline* the vase. By measuring the width of the outline, we can compute the midpoint of the contour, and fix our assumed line of symmetry. The next step is to split the original outline contour through this vertical axis we have assumed, in order to obtain two contours. Our aim is to choose one side contour, reflect it, and then connect it to the original side, so that we can have a symmetrical vase outline. The chosen side was selected either because of a pre-set variable (for example, if we always pick the *left* side contour) or in other cases, for instance in very poor quality images, the side with fewer points[88] along its contour, in a certain range.

Before creating a complete symmetrical outline, the chosen side contour must first be smoothed. To smooth these side contours, we employed a three-step process:

1. For the points in the main body of the vase (i.e., excluding the base and mouth of the vase), we restricted the points in the side contour to include the maximum / minimum (depending on whether the right side or left side is chosen) x-coordinate that appears for every unique y-coordinate. This decreases the number of points in a contour and hence, helps de-noise the contour.

2. Remove contour *indents*. As the vases in our sample did not exhibit the most complicated of forms, we assume that any *fractures* or indents in particular areas of the outline (for example, the main *body* in the centre) are deformities caused by external factors such as a fault in the binarization or contour extraction steps. Broadly, we define these indents as points on a contour that are preceded and

[88]The assumption here is that this is the least noisy side.

succeeded by points that have an x-value that is greater than (or less than, for left-side contours) the x-value of the point in question.

3. Smooth the main body of the side contour using B-splines[89], based on cubic splines. Cubic splines can be used to *smooth* a curve, by fitting multiple cubic polynomials onto a curve, joined at various points, called knots. Meanwhile, B-splines (where the name is shortened from *basis*-splines) allow us to express the cubic splines by using a linear combination of the basis functions of order 3, with respect to the *knots*. As described in (Höllig and Hörner, 2013), this enables us to describe a smooth curve with a linear combination of B-splines, S:

$$S = \sum_k c_k b_{k,\zeta} \tag{4.1}$$

where the degree $n = 3$ in our case, and $\zeta = (\zeta_k, \cdots, \zeta_{k+n})$ is the vector of knot positions. The c_k are called the *control points*, that are used to define the B-spline functions. By employing B-splines, computations are numerically more efficient than simply incorporating traditional splines, whilst allowing greater freedom to control smoothness, and the positioning of knots (for example, by utilizing uniformly-spread knots). Moreover, by fixing the parameters in this B-splines interpolation step, the process of contour smoothing is automated and hence the processing of the contours is more efficient. For more details on curve fitting with B-splines, we refer the reader to (Höllig and Hörner, 2013) and (De Boor, 1978).

Once the side contour has been smoothed, a duplicate is created, which is then reflected in the y-axis and subsequently connected to the original side contour. Thus, for every outline curve extracted from an image, we obtain a symmetrical contour.

Procrustes Alignment

At the final stage, outlines are processed to remove global shape-preserving transformations such as scaling, using Procrustes alignment. As detailed in the introduction of Chapter 2, Procrustes analysis involves optimally aligning a shape to a template using translation alignment, rotation, scaling, and reflection. Here, the template was chosen arbitrarily from our original sample. This was followed by a reparametrization of the curves so that they all have N points. In this project, we set $N = 139$. This number was

[89]namely with scipy's[90]`scipy.interpolate.splrep` and `scipy.interpolate.splev` functions

[90]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splev.html

chosen after numerous tests, based on the fact that at this number, we could not see any noticeable difference between a linearly interpolated curve and the *true* outline.

Although Procrustes alignment is not necessary when incorporating our primary shape analysis methods for the classification, its usage is required for other methods included in our tests. Therefore, as we make comparisons between various methods, to perform as a fair a test as possible, Procrustes alignment becomes a vital part of our data processing.

4.1.7 Quantifying Vase Shape Variation

How different is an amphora from a krater?

Our objective is to use shape analysis to classify a sample of vases. Though the shape / form of ancient vases has been mathematicially evaluated in the past, for instance, using Elliptic Fourier Analysis to study ancient Chinese pottery in (Wang et al., 2021), a comparative analysis of their use in classification is largely absent. Additionally, when shape methods are applied in classification-related projects, the focus tends to be either on one specific method, or on a single groups of methods, such as geometric morphometric methods (as seen in (Sheets et al., 2006), a zoological study of feather shapes, which compares various morphometrics methods). For that reason, in this project we work with a selection of different mathematical methods including the most common method used in historical classification, and methods of diffeomorphic shape analysis. These mathematical methods will be used to compute differences between pairs of vases. Subsequently, they will be used as the input in a machine learning classification algorithm. In this section we briefly outline the shape analysis methods that we incorporate in this study, and refer the reader to the Background Chapter 2 for a more detailed overview.

Geometric Morphometrics

The name *Geometric Morphometrics* habitually appears in archaeological literature relating to shapes. Simply put, as discussed in Section 1.2.1, geometric morphometrics are landmark or semi-landmark approaches to analyse morphological variability in shapes. These methods can be used to visualise changes in the shapes of objects, hence their importance in archaeological classification, as emphasised in (Selden Jr, 2019). Mathematically, one standard approach can be broken down into four steps. Though this can go by various names, in this project, we call this method Eigenshape Analysis.

Eigenshape Analysis:

1. Find landmarks outlining the form of an object, in each object. $S_{\text{landmarks}} = \{c = (p_1, \cdots p_N) \subset \mathbb{R}^2\}$ where the N landmarks are represented with coordinates, for example, $p_i = (x, y)$.

2. Use Procrustes analysis to remove variations in the similarity transforms (i.e., the shape-preserving transformations of scaling, rotation and translation).

3. Find a linear transformation with Principal Component Analysis to reduce the dimension of the data using the top $k < 2N$ components.

4. Compute distances with the Frobenius norm between all pairs of shapes in the sample, where the shapes are vectors of size k.

For the majority of this project, we refer to the outlines of our vases as curves. However, unlike the other methods, Geometric Morphometrics works on landmarks not curves. As these curves describe the coordinates of the positions on the image that correspond to the object's outline, we can consider them as we would a set of landmarks[91]. Here, we'll call this a semi-landmark approach[92], where we not only have an equal number of evenly spaced points on each curve, but we also have the first point on all curves, positioned at the same position (namely, on the top centre of the vase outline).

Shape Analysis Methods

Previously, we defined the shapes in our sample as outlines of vases, post-Procrustes alignment. Thus, it would be more natural to consider these outlines as curves rather than treat them as landmarks, as Eigenshape Analysis does. For this, we turn to diffeomorphic shape analysis.

The first thing we note is that though we may be visualizing vase outlines as curves in $\mathbb{R}^2$, it is not as straightforward as it may seem. In the case where each curve is described with N points, the shape space of our curves is in actual fact a submanifold

[91]We reiterate that these points in our dataset are not *real* landmarks, as apart from the start and end points at the top and base of the vase, there is no correspondence between the points across the set of images.

[92]Though note that we are not referring to the official term of *semi-landmark* which is traditionally used to describe the *sliding landmark* method, described in (Gunz and Mitteroecker, 2013).

of $\mathbb{R}^{2N}$. Moreover, as discussed in Chapter 2, the shape space can admit a Riemannian metric. Unfortunately, such Riemannian metrics in the shape space are not so simple, in computational terms. A solution to this could be to transform the shape space into a more *manageable* space. We look at three methods which utilize diffeomorphic (*elastic*) shape analysis to quantify differences between shapes; two of them transform the shape space, whilst one of them remains in the original, rather complicated, shape space, as illustrated in Figure 2.5. More details about all three methods can be found in the Background chapter but we provide a brief outline here for continuity.

LDDMM: We can quantify differences between the shape of two vases by deforming the outline curve of one, through bending and stretching it, so that it resembles the other. The energy utilised to compute this deformation is the desired *distance* quantity. This is the basic foundation of the LDDMM (Large Deformation Diffeomorphic Metric Mapping) framework. More explicitly, we look for a geodesic between two curves, as described in Section 2.3.1, with the geodesic energy described in Definition 2.16. We compute this *energy*[93] between all pairs of vases in our sample. Here, we use the implementation described in (Marsland and Shardlow, 2017), with the stochastic parameters set to 0, and the time-step parameter set to 20.

SRVF Path-Straightening: In this method, we transform the shapes into a simpler space by employing the square root velocity function (SRVF) representation of the shapes, instead of directly using the original curves.

$$q(t) = \frac{\dot{c}(t)}{\sqrt{||\dot{c}(t)||}}, \tag{4.2}$$

for a curve $c(t)$, where $||.||$ is the standard Euclidean norm in $\mathbb{R}^2$. As we saw in Section 2.4, to compute distances between two shapes, the SRVFs of the shapes and its tangent vectors are projected into the shape space and tangent space respectively. The process then involves straightening a path between the two shapes in κ steps[94], by incorporating a gradient descent approach, until the path is a geodesic. The length of the geodesics show

[93]Note that this *energy* equation involves an approximation of a derivative of the curve Φ. As we will see later on in this chapter, inaccurate discretisations of the curve and hence approximation of this derivative will affect the results of distances between vase outlines.

[94]We note that after a few experiments, we decided to set $\kappa = 2$ as we found that at this value, distances were sufficiently unchanged (in comparison to greater values) whilst maintaining a fairly quick computation speed.

us how much energy is required to morph one shape into another. More details about the SRVF Path-Straightening method can be found in Section 2.5.1, and the implementation used here can be found in the fdasrsf[95] Python library.

Geometric Currents: Another method which involves transforming the original space of shapes uses currents, based on Geometric Measure Theory, (Benn et al., 2019). Here, the equivalence class of a curve is mapped under the current map into a single point in $\mathcal{H}$; consequently, we can compute straight-line distances in the normed vector space. As this space is equipped with a Euclidean metric, distances and other statistical techniques such as PCA can be easily computed (see Section 2.3.2). Note that there were three parameters involved in this implementation, based on the matrix-size, the mesh-size, and a scaling parameter. For more details on these parameters, and the implementation we incorporated here, see Section 2.3.2 and the paper (Benn et al., 2019), which provides a thorough account of the geometric currents method.

Overview of Methods

To date, shape analysis methods have seldom been used in real-world classification studies. We want to compare these shape analysis methods with the most common method used in this area, which is why the Eigenshape analysis method features here. Overall, we have chosen four different methods to obtain distances between our vases. One method uses a semi-landmark approach (Eigenshape Analysis) whilst the other three treat the outlines as curves. Importantly, this means that the shape analysis methods do not require the points to be in correspondence. Moreover, two of the methods (SRVF Path-Straightening and Geometric Currents) are invariant under orientation-preserving reparametrizations. As discussed, in order for the experiment to be as fair as possible, all vase outlines have been reparametrized to have the same number of points and they have been Procrustes-aligned to remove variability of orientation-preserving (and shape-preserving as a whole) transformations.

Our four methods certainly have their differences, whether it's mathematically or computationally. Eigenshape Analysis transforms vase outlines into N-size vectors based on the top N principal components. Meanwhile, Geometric Currents incorporates measure theory and elastic shape analysis to *transform* a curve into a single point. On the other

[95] https://fdasrsf-python.readthedocs.io/en/latest/

hand, SRVF Path-Straightening and LDDMM methods search for some optimal deformation between shapes. However, despite their differences, one shared attribute is that they can all output a *distance*. We therefore obtain four distance matrices (one for each method) showing the pairwise distances between all pairs of vase contours.

4.1.8 Machine Learning Classification

Our goal is to compare the four chosen mathematical methods with respect to their classification performance. To classify the vase outlines, we incorporate a machine learning algorithm. Most often, machine learning and data mining algorithms require features as inputs, in our case however, the sole input of our chosen classifier will be a distance matrix. This prerequisite automatically reduces the options for machine learning classifiers to use in this project[96]. Out of the few options that remain, one algorithm is particularly efficient for what we need: a k-nearest neighbour classifier.

A k-NN Classifier

The k-nearest-neighbour, k-NN, algorithm was first developed by the statisticians Evelyn Fix and Joseph Hodges in (Fix and Hodges Jr, 1952). Broadly, the algorithm works by classifying an object based on the majority classification of the object's closest neighbours. Commonly, a Euclidean distance is used to find the *nearest* neighbours to an object. However, instead of inputting original curves from a sample and employing an arbitrary distance metric, the algorithm can be easily modified to use a distance matrix as its input instead. In this project, we implemented our own k-NN classifier, a brief outline of which can be seen in Algorithm 9. Not only does this algorithm require a distance matrix as its input, but it also tests multiple classification results based on differing values[97] of k. The

[96]Note that though standard deep learning methods can't be used, a neural network can be trained to find distances between **pairs** of curves, as in (Hartman et al., 2021), where a Siamese convolutional neural network was trained on square root velocity distances. We tested this classifier on our dataset, but we obtained poor results and thus we did not pursue it further. One reason for this poor performance can be attributed to the relatively small training set, as traditionally, deep learning methods require large datasets. Hence, with a larger dataset, this could be a potential future avenue to explore.

[97]Various values for k were tested, with an example shown in Figure 4.11. These values were based on the minimum number of training data per class. In our specific dataset of Greek vases, the number of training data per class ranged from 12 outlines, to 25, thus our k-NN algorithm tested integers up to 12. From here, the classification results that provided the highest *score* were output, as explained. We remark that this method of selecting the *best* k is indeed not a robust method that should be used

classification that resulted in the highest score is the final output. In order to score the classifications, we opted for a F_1 score. We use this score as it is highly-popular scoring method within classification studies. It is based on the harmonic mean of the precision and recall scores, denoted by P and R respectively:

$$P = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Positives}|} \tag{4.3}$$

$$R = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Negatives}|} \tag{4.4}$$

The F_1 score is a variant of the more general F_β score that uses a weight term β (which we will discuss in Section 5.1), with β set to 1:

$$F_\beta(P, R) = (1 + \beta^2) \times \frac{PR}{P(\beta^2) + R}$$

$$F_1 = F_{\beta=1}(P, R) = 2\frac{PR}{P + R} \tag{4.5}$$

We note that these definitions are based on binary classification. Therefore, the F_1-scores are computed for each class, against all other classes, and combined together with a weighted average:

$$F_1^{multiclass} = \frac{1}{N}\sum_{i=1}^{N} N_i(F_1)_i \tag{4.6}$$

where N is the total number of samples, N_i is the total samples in the i^{th} class, and $(F_1)_i$ is the F_1-score for the i^{th} class (in this case, the binary classification is based on whether a sample is of class i, or not, and so on). For the remainder of this thesis, we refer to $F_1^{multiclass}$ defined in Equation (4.6), as the F_1-score.

Boostrapping

In general, classification depends on how a dataset has been split into *training* and *testing* sets. For small datasets in particular, there is a possibility that the mere positioning of samples in the two sets can strongly affect the final classification results. We took this into account, and opted for a bootstrapping approach, creating 100 training and testing

for general classification analyses. Instead, it was used solely for the purpose of comparing our different methods, with their *best* possible results. Some examples of classification results with different values of k can be seen in Figures 4.10 and 4.11. Here, unless stated otherwise, we will plot these *best* results, with k ranging from 3 to 12. In general, if a k-NN classifier is desired on particular classification analysis, k should indeed be fixed (for example $k = 5$ in our further vase classification analyses in Section 4.2 of this chapter).

Algorithm 9: Distance Matrix k-NN

```
new_knn(D,class_data,train,test,kNeighbours=np.linspace(3,12,10))

Aim:
k-NN classification using a distance matrix, D.
Input:
Pairwise distances between shapes, D, and class information from the training
(train) and testing samples (test), class_data.
Code:
  1. Create two dictionaries, train_details and test_details, and append to each of
     them, the names of the samples in the training set and testing set respectively,
     along with each sample's true class, seen in class_data.
  2. Let top_score=0 and begin k-NN process:
     for $k \in$ kNeighbours do
         pred_details = {}
         for test_sample ∈ test do
             • Find all distances in D, between test_sample and samples in train.
             • Sort distances in ascending order to find the training samples that
               correspond to the k smallest distances.
             • Let k_classes be the set of classes, of those k training samples.
             • pred_class = mode(k_classes) and save to pred_details
         end
         Compute $F_1$ between the true classes (test_details) and predicted
         classes (pred_details) using Equation (4.6).
         if $F_1 \geq$ top_score then
             top_score = $F_1$
             classification = pred_details
         end
     end
return classification - the classification scores of the samples in test.
return top_score - the top $F_1$-score computed.
```

sets, where the objects were quasi-randomly[98] selected from the original data sample, before proceeding with the k-NN step. After each of the 100 iterations, for each distance matrix, the algorithm outputs the top F_1-score. Finally, after all iterations, the average of all 100 classification scores is computed for each of the distance matrices. This is then the final output of the algorithm, along with the top classification results per method, per iteration. We provide an example of this algorithm in Algorithm 10.

[98]Our tests are multi-class classifications, thus it is important to assure that all classes appear in the training sets. To do this, training and testing sets were created by sampling the data by class. For each class, a proportion (denoted by `p` in Algorithm 10) of data-points were randomly selected for the training set, and the rest were labelled as the testing set.

Algorithm 10: Bootstrapping k-NN Example

```
bootstrap_knn(distance_matrices,index_data,N=100,p=0.25)

Aim:
$N$ bootstrapping tests with a $k$-NN classifier (Algorithm 9) on distance matrices.
Code:

  1. Create N training and testing sets, train_test. Let $i = 0$.
     while $i < N$ do
         training_set, testing_set = []
         for $c \in$ classes do
             • Let names contain the list of samples, with class c.
             • train = random.sample(names,int(length(names)*p)) and
               let test be the samples in names that are not in train.
             • training_set.extend(train), testing_set.extend(test)
         end
         train_test.append([training_set,testing_set])
         $i \mapsto i + 1$
     end

  2. Let avg_scores, all_scores, all_classifications = [] and run $k$-NN:
     for D ∈ distance_matrices do
         scores, classifications = []
         for (train,test) ∈ train_test do
             top_$F_1$,predictions = new_knn(D,index_data,train,test)
             scores.append(top_$F_1$),
             classifications.append(predictions)
         end
         avg_scores.append(average(scores))
         all_scores.append(scores)
         all_classification.append(classifications)
     end

return all_classification - the classification scores of the samples in test.
return all_scores - N top $F_1$-scores computed for every distance matrix.
return avg_scores - average top $F_1$-score computed for each distance matrix.
```

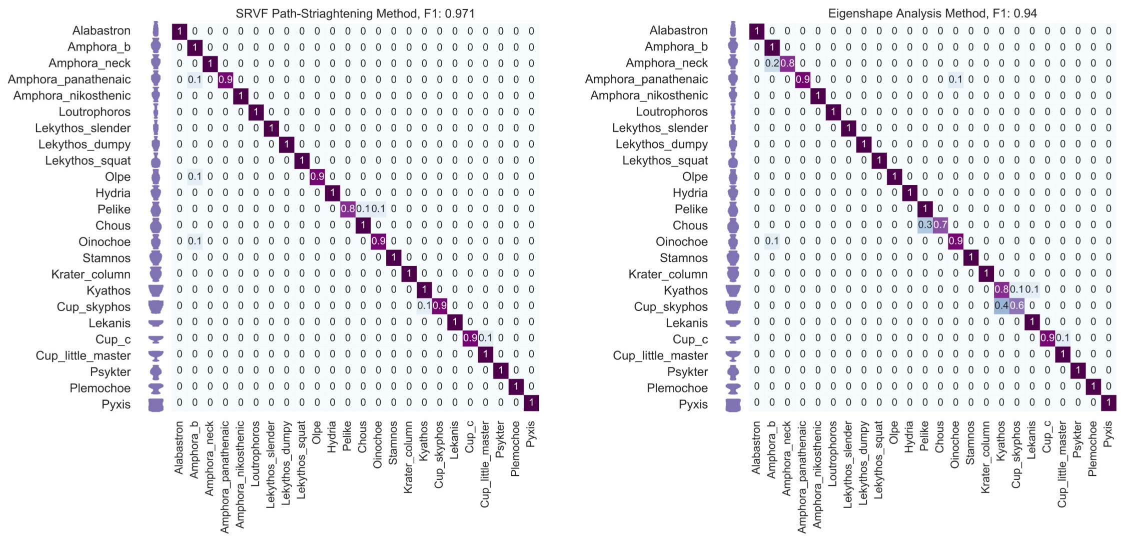


Figure 4.9: Confusion matrices showing the classification results from a k-NN classifier on a distance matrix computed with the SRVF Path-Straightening method (left) and the Eigenshape Analysis method (right). The same training and testing sets were used for these results. For this particular training and testing set, these two methods outperformed the other two, with the SRVF Path-Straightening on top.

4.1.9 Classification Results

Across the various methods in this project, there are many parameters that require selection, whether based on sufficient accuracy of the classification, such as k in the k-NN classification or the training proportion parameter, `p` in the k-NN algorithm, 10, or on the shape methods themselves, such as the number of principal components to use for the Eigenshape Analysis method, or the three parameters required in Geometric Currents. In order to *optimize* these parameters, we took on an experimental approach which involved running multiple k-NN tests on various parameter options. Here, we plot the results from some of these classification tests. Furthermore, in this section, we plot some of the top classification results that we obtained.

Overall, when studying the classification results from the k-NN classifier, we found that the SRVF Path-Straightening method consistently outperformed the other methods. This can be seen in the Table 4.1.

[99] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

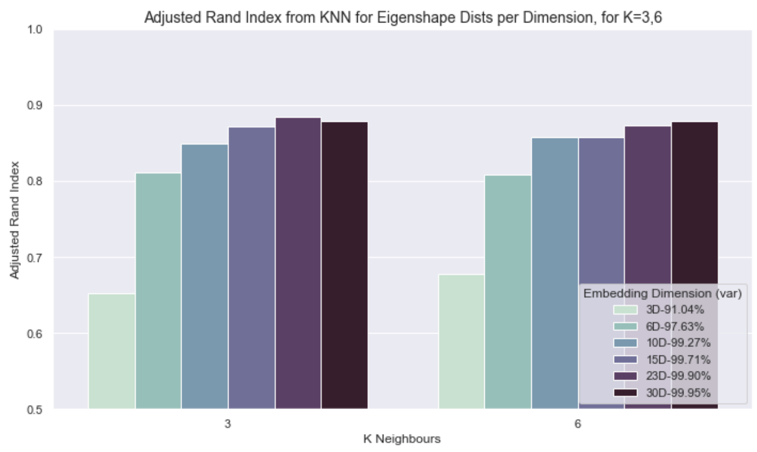


Figure 4.10: Eigenshape dimension tests. In order to compare the effects of altering the dimensions in Eigenshape Analsyis, we implemented a k-NN classifier (shown here for $k = 3, 6$) on various distance matrices. This time to make the comparisons, we computed the average Adjusted Rand Index[99]. We found that though the majority of the variance was held in just a few principal components (for example, $> 90\%$ in a mere 3 principal components), we still had to increase the number of dimensions rather drastically in order to significantly improve the results. The accuracy started to stagnate at approximately $N \geq 30$ with 99.95% of the variance covered, and this was also mirrored when adjusting the neighbourhood value k.

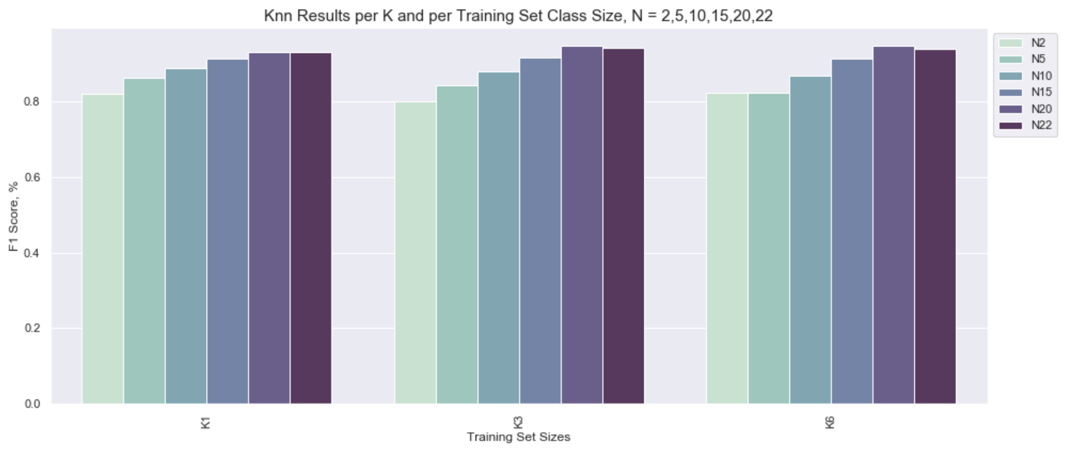


Figure 4.11: We plot the average F_1-scores from k-NN tests (shown with $k = 1, 3, 5$) but using various values for the proportion parameter p (i.e. the parameter used in Algorithm 9 that determines the number of samples that are put into the training set). We tested a variety of values for p, where $2 \leq \texttt{p} \leq 22$. Interestingly, though the results do improve when p increases, the average F_1-scores at even the lowest values for p are still very respectable.

Method	SRVF Path-Straightening	Eigenshape Analysis	Geometric Currents	LDDMM
Average F_1	0.967 ± 0.002	0.920 ± 0.003	0.916 ± 0.003	0.909 ± 0.003

Table 4.1: Table containing the mean average top F_1-scores $\pm 95\%$ confidence intervals, from 100 k-NN boostrapping tests. Statistical t-tests showed that the F_1-scores resulting from different methods were all statistically different from each other, except for the average F_1-scores computed using Eigenshape Analysis and Geometric Currents, which were found not to be statistically significantly different.

From Table 4.1 we learn that the distance matrix computed using the SRVF Path-Straightening method results in the best classification results using a k-NN classifier, compared to the other three methods. The F_1-scores resulting from the Geometric Currents approach and Eigenshape Analysis are very similar, with the results of a standard t-tests (with 5% and 1% thresholds) showing p-values to be $p < 10^{-6}$.

Another interesting result seen in Table 4.1 is the rather poor performance of the k-NN classification results based on the distance matrix from the LDDMM method. Although LDDMM is the only diffeomorphic method here that worked in the original shape space of the vase outlines, the metric uses a trade-off between the precision of the transformation and the length of the geodesic path. The smaller the trade-off, the more processing power that is required to make the computations. This, along with the potential presence of noise, could be the reason why it didn't perform as well as the other diffeomorphic methods that, unlike LDDMM, work in much simpler shape spaces.

Expert Analysis

At the start of this section we asked three questions that shape our vase classification project. One question asked whether an algorithm can classify vases as well as a human would. This is an important question as vase classification which, from centuries ago, to modern-day analysis, has relied on the opinions of the experts. The average person may be able to pick out an amphora from a set of cups, but the task is not always so simple. Vase shapes differ varyingly, and in some cases, the differences are incredibly subtle. However, Greek-vase experts have been trained to notice these differences, irrespective of

[99]This score is similar to the computation of *accuracy* (proportion of correct predictions).

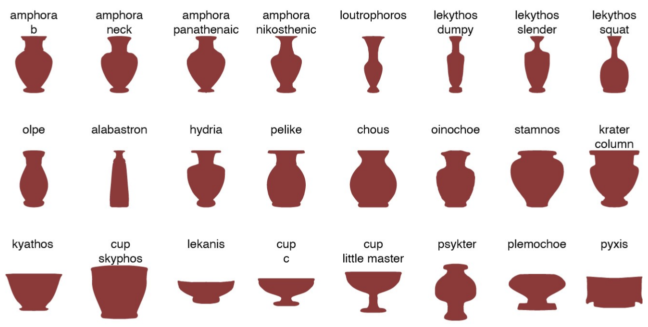


Figure 4.12: Examples of the expert's vase sample. The outline contours were filled and scaled to be the same size (through the Procrustes-alignment stage of the contour extraction process). This sample was then sent off to the vase experts in order for them to perform their clustering. Here, we display one filled vase outline for each of the classes in our sample.

the subtleness, and utilise this information to classify vases.

Expert vase classification almost always requires the expert to have the vase at hand, or at minimum, a high quality image of the vase. This way, they are provided with all the information that a vase can offer, to make their classification; from vase textures, and materials, to the vase paintings and possible potters' signatures[100]. On the other hand, our methods used just the *shape* of the vase to make a classification. Thus we wondered how well experts can classify vases if they too could only look at the shapes, and whether shape was indeed *enough* to perform meaningful classifications.

In order to answer such questions, we created another training and test set of vase outlines. Both of these sets were given to our k-NN classifier so that we could use our four methods to obtain a classification of the outlines in the test set. Meanwhile, three expert scholars of Ancient Greek vases were given images of vase outlines from the test set (examples of which are shown in Figure 4.12). The experts were told how many different vase classes there were, but were not given the names of the classes. From here, they were tasked with partitioning the set of outlines into groups. Finally, we assigned a predicted class label based on the experts' clusters (for instance, if the majority of amphorae were put into the first cluster, then cluster 1 would be labelled as amphora, and so forth), and used this to compute F_1-scores.

[100]Potters' and painters' signatures often appear on ancient Greek vases. See (Immerwahr, 1984) for a study connecting painters to potters by their signatures.

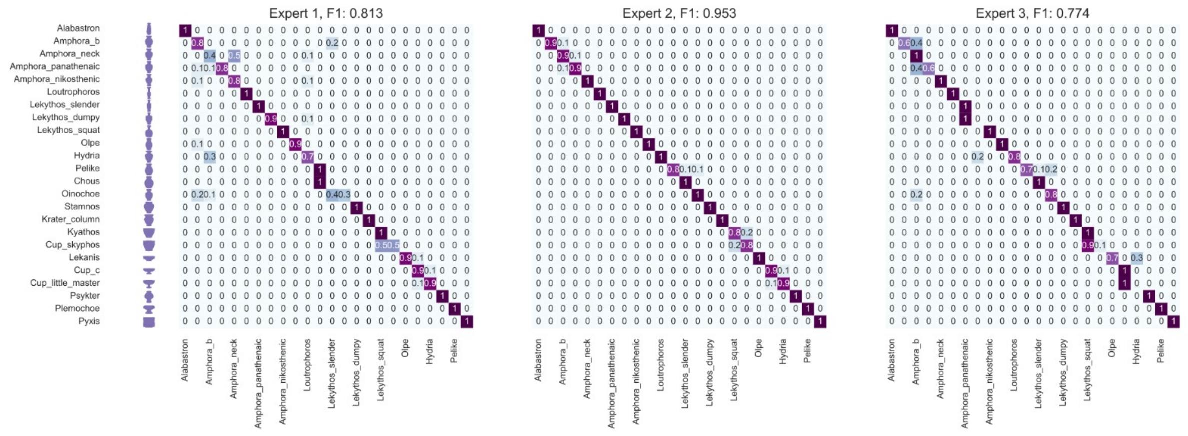


Figure 4.13: Confusion matrices based on the clustering results from our three vase experts. The clusters were assigned a class, based on the majority class that appeared within them. Subsequently, confusion matrices were plotted and F_1-scores were computed. Though this approach is (understandably) not often used to portray results from clusterning analysis, we adopted this technique in order for us to be able to make straightforward comparisons between the expert's clustering results and the machine learning classification results. The F_1-scores for each expert are displayed in the title of the plots.

Figure 4.13 shows the confusion matrices from the experts' clustering results. Two out of three of the experts gained an F_1-score lower than any of the four methods on the same test set (the confusion matrices from two of the methods can be seen in Figure 4.9), emphasising the worthiness of algorithmic methods for vase classification. One expert performed tremendously well and outperformed all methods except the SRVF Path-Straightening method. When taking a deeper look at the experts' confusion matrices in conjunction with the confusion matrices from our k-NN classifier, we see that there are certainly similarities between the mistakes that are made. For example, there seems to be confusions between pelike and chous vases (see the 12th and 13th rows/columns, respectively, in Figure 4.13) in all of the confusion matrices by experts and algorithms. Experts can spot small differences between shapes of vases. Thus even when it comes to vases with incredibly similar shapes, such as skyphos and kyathos vases (see Figure 4.14, or the 18th and 17th rows/columns in Figure 4.13), the experts do well to classify at least half of the outlines correctly. With these two vase types in particular, one method that performed extremely well at distinguishing between the two was, once again, the SRVF Path-Straightening method.

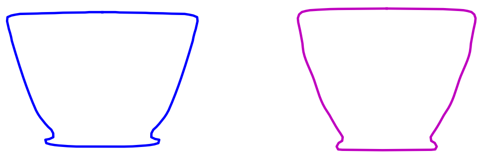


Figure 4.14: *Can you spot the difference?* Outline of a kyathos (L), and skyphos vase (R). The shapes of these two classes of vases are often very similar, with the main differences appearing in the base. Therefore it is understandable that these two were one of the pairs of classes that the vase experts confused the most. The SRVF Path-Straightening method classified the vases of these two classes almost perfectly, as illustrated in Figure 4.9, which shows the same testing set that the experts used.

4.1.10 Means & Principal Components

Previously, we saw that the top method for vase classification was not in fact the traditional morphometrics method used in archaeological (and anthropological) shape studies, Eigenshape analysis, but instead the diffeomorphic method from elastic shape analysis, namely, SRVF Path-Straightening. This is the motivation for further comparisons between diffeomorphic methods and traditional methods.

The Average Vase

Traditionally, a mean shape can be computed from a set of landmarks (that are in alignment) by computing the standard mean average of each point. In diffeomorphic shape analysis, a related average can be computed, known as the Karcher mean, as detailed in Section 2.5.3 of the Background chapter. Here, to represent our diffeomorphic methods, we can use the diffeomorphic *distance* metric used in our SRVF Path-Straightening method for example, and compare this to the standard (pointwise) Euclidean mean.

Computing the mean shape is a useful tool in shape analyses; not only does it give us an insight into what our data looks like, but the means can also be used as templates, for example for further classification. In Figure 4.15, we see a plot showing the (filled) mean outline for a subset of vase classes from our sample. On the left, the average shapes were created using the Karcher mean, whilst on the right, we used the standard Euclidean mean. Though they may look similar from afar, when looking more closely, we see that

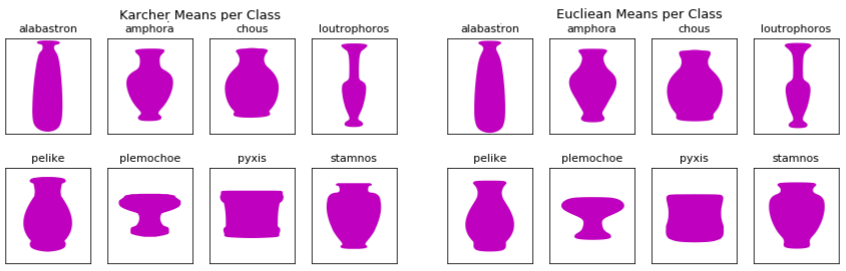


Figure 4.15: Average shapes. The mean shape of vases for a few classes featured in our sample. In the left plot, the average is computed using a Karcher mean, whilst in the right plot, a standard Euclidean mean is used. Here, we see that the Karcher mean has performed much better in retaining the shape information than the standard mean. This point is particularly clear in the pyxis vase class, where the standard Euclidean mean has ignored the true lip and base structure of the vase, unlike the Karcher mean equivalent.

the Karcher means provide much more information that the Euclidean means. This is particularly evident in the bases and lips of the vases. An extreme example for this observation is the pyxis vase, where the standard mean has missed the shape of the base and lips entirely. These plots highlight the advantages that diffeomorphic shape analysis has over traditional methods when studying shapes of objects.

Plotting Principal Components

Another common tool used in analysing shape data is principal component analysis (PCA), which can be used to visualise the samples in a dataset. By plotting the top principal components (defined by the majority percentage of variance that the components represent) from a dataset , we can examine whether any possible clusters exist.

In order to compare traditional PCA with our shape analysis methods, we obtained principal components from the Geometric Currents method, and the SRVF method. As the Geometric Currents method transforms the shape space into a standard Euclidean vector space, PCA can be performed in the traditional way. Meanwhile for the latter method, principal components can be computed by considering the variation of points to the Karcher mean, using the implementation of Tangent PCA described in Section 2.5.3.

Figure 4.16 shows principal components plots using the two diffeomorphic methods (left and centre), as well as traditional PCA (right). Note that these plots illustrate a better,

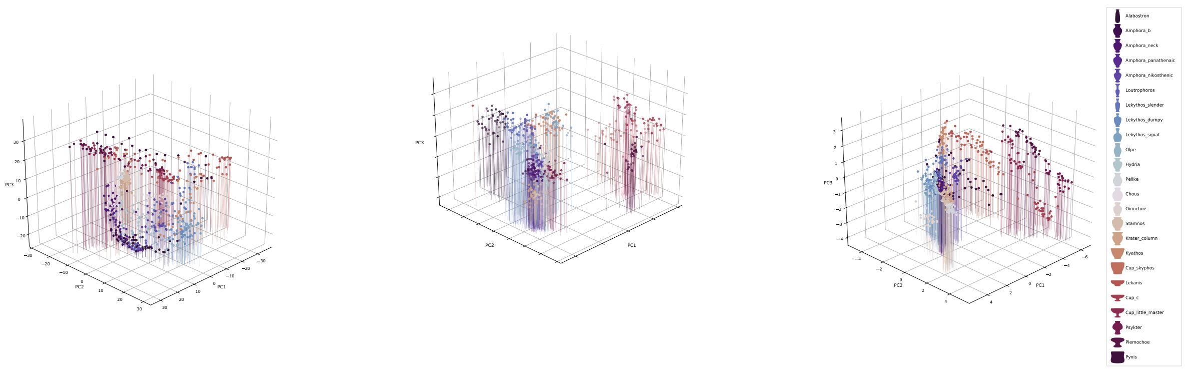


Figure 4.16: Principal components of the shapes of vase outlines, where the top three principal components represent $> 90\%$ share of the variance. On the far left we have the first three principal components computed using tangent PCA (tPCA); in the centre we have the Geometric Currents PCA result; and on the far right we have the standard PCA results, for all vases in our sample. These plots show us that the two methods of curve shape analysis perform better in clustering the vase outlines.

more defined, separation between classes when using the curve-based methods. This further emphasises the benefit of the inclusion of shape analysis methods in studies of object shapes.

4.1.11 Repeating the Method on Other Datasets

Throughout this section, we have been analysing and comparing traditional methods used to study shapes with methods from diffeomorphic shape analysis, to study and classify shapes of ancient Greek vases. Firstly, we have learnt that classification on shape alone can certainly yield successful results. More importantly, this project has shown that our recipe for classification not only works, but performs very well. The recipe being:

$$\text{Contour Extraction} + \text{SRVF Path-Straightening Distances} + k\text{-NN}.$$

The vases seen in this project came in all sorts of shapes, but they are certainly not the most complex of shapes. In order to scrutinize our *recipe*, we ran the same experiment on different classes of shapes, that were not only arguably more complex, but they also contained classes that were far more similar than the closest classes that we saw in our vase dataset[101]. These were leaf outlines from the Swedish Leaf Dataset[102][103] and gastropod shells. As before, we consulted three experts for each dataset, who in turn provided their own clusterings (which were subsequently turned into *classifications*).

When testing our methods on these two new datasets, we found that the results echoed those from our vase project, with the SRVF Path-Straightening method outperforming all methods and experts, followed by the similar results from Geometric Currents and Eigenshape Analysis, which were trailed, once again, by LDDMM. The confusion matrices from all of the tests can be found in Figure 4.18. Moreover, what was even more interesting about these results, was the large difference between the scores of the SRVF Path-Straightening method, compared to the classification score based on the top experts. For example, let's consider the gastropod shell dataset. Many of these classes are incredibly similar to each other, as seen in Figure 4.17, which shows the outlines from the test set. This is particularly true for the cone shapes (*Conasprella*, plotted in green, and *Conus* plotted in cyan), as well as other shell genera such as the *Buccinum* (blue) and *Neptunea* (red). Therefore, it is not too surprising that even the top classification results from our malacologist failed to distinguish between these classes, as shown in the bottom right confusion matrix in Figure 4.18. On the other hand, the k-NN classification resulting from the SRVF Path-Straightening method performed remarkably well in distinguishing the shapes of these very similar classes, and hence the (greater than) 30% increase in F_1-scores, in comparison to the top expert's score. These results simply accentuate the great potential of elastic shape analysis methods (particularly SRVF Path-Straightening), in the classification of object shapes, no matter how complicated.

[101]For example, the average shapes of the *kyathos* and *skyphos* vases are less similar, than the average shapes of the *Conasprella* and *Conus* shells (see Figure 4.17).

[102]`https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/`

[103]Another reason for incorporating the Swedish Leaf Dataset is because previous tests in shape analysis literature also utilize this dataset, such as (Laga et al., 2014).

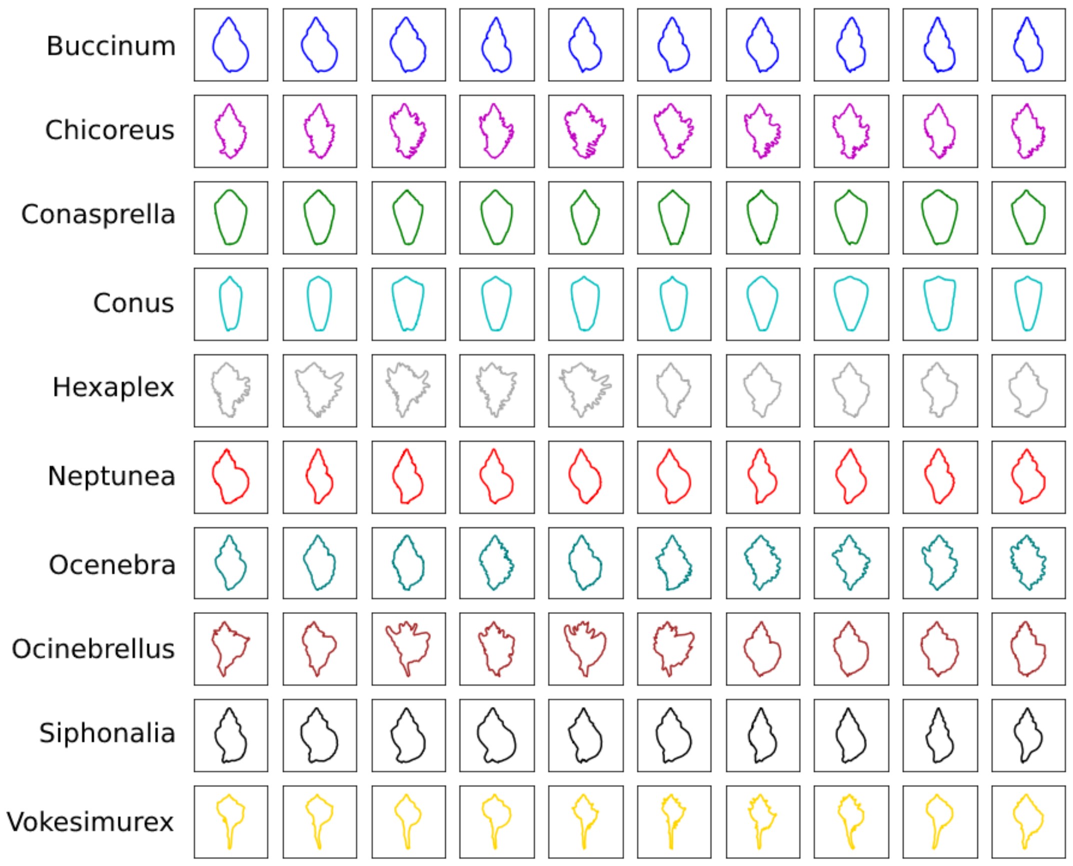


Figure 4.17: Outlines of gastropod shells from the test set that was used in our classification and clustering analysis. Here, we can see just how similar some of the shell shapes are. We can also see how complicated the shapes of some of the species are, such as the *Chicoreus* (magenta) and the *Hexaplex* (grey).

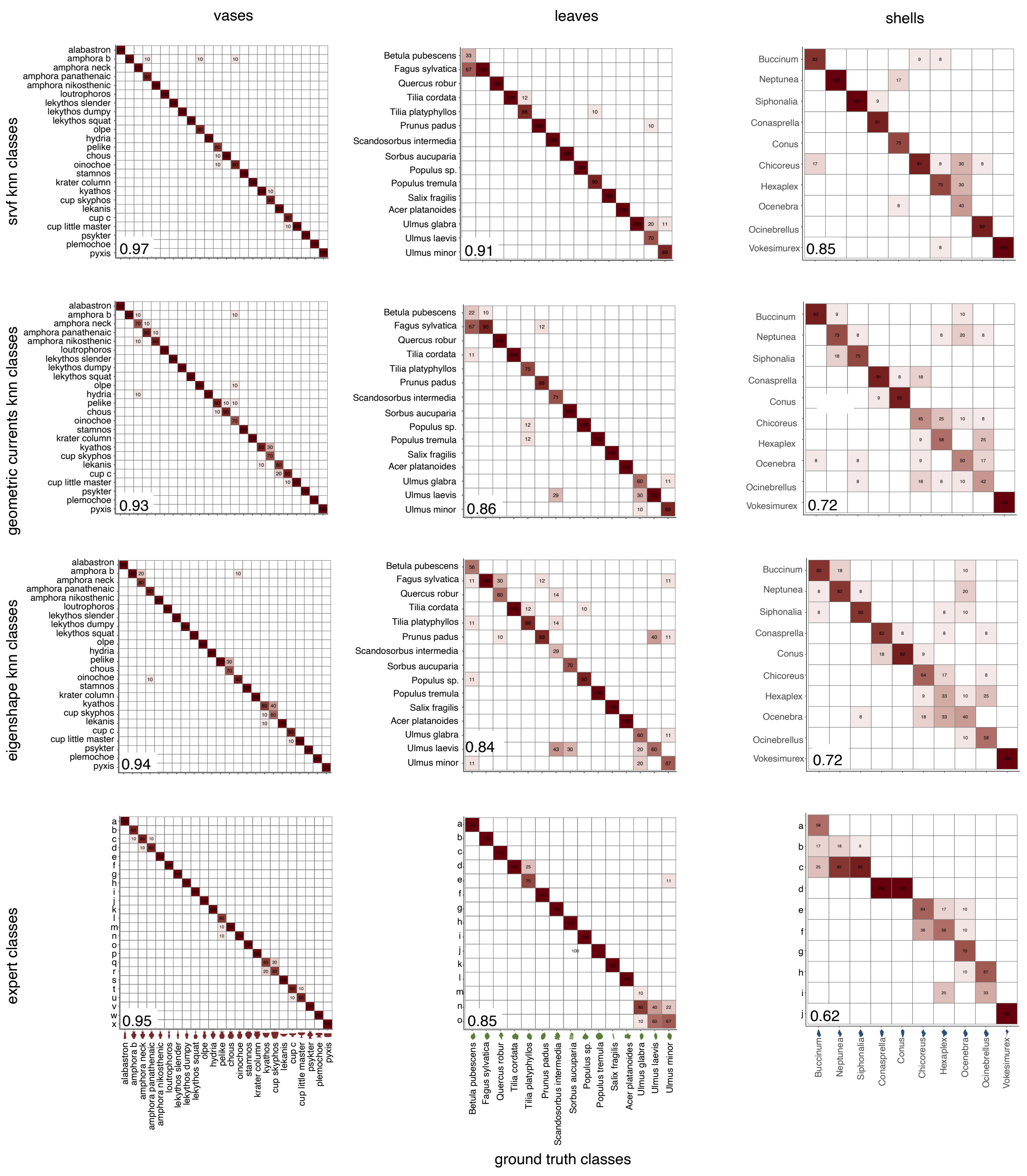


Figure 4.18: The top three confusion matrices resulting from k-NN classification on the distance matrices computed with the mathematical methods of SRVF Path-Straightening (top row), Geometric Currents (top centre), and Eigenshape Analysis (bottom centre), and the experts' classification (bottom). The results are shown for all three datasets: ancient Greek vases (far left), leaves (centre), and shells (far right). The F_1-scores are shown on the bottom left corner of each of the confusion matrices. The results here emphasise how well methods of shape analysis perform in classification.

4.1.12 Overview

Future Work

Throughout this project, we have seen that methods of elastic shape analysis have performed well in the analysis of closed outlines from images. In particular, the SRVF methods have constantly come out on top. Recall from Section 2.4.2, that the metric used in this SRVF framework, is broadly based on some *alignment* between the shapes. However, we note that in this case, closed curves lose rotation along the curve. For example, if we consider the outlines of two shells, it is possible for one outline to be *matched* to a rotated version of the other. In other words, there is no alignment of *structures*. Depending on the questions asked, this may cause issues in certain datasets. However, as we have a more general interest on the shapes of objects, particularly in classification, it is not a concern. Though in the future, we can work on a version of this diffeomorphic method that enables us to *fix* a desired number of points on the curve, in the alignments.

There is a plethora of datasets containing images of ancient Greek vases that have yet to be fully explored. In this project, we have seen just how well we can classify vases by simply studying the shapes of their outlines - in particular, by using tools from elastic shape analysis. Our end goal is to use these methods to analyse much larger datasets of vase images. Henceforth, the next steps are to improve the handle removal code (to speed up the automatic contour extraction process) and focus on SRVF Path-Straightening, so that we can begin to discover more about the shapes of these ancient vessels.

Summary

Our underlying aim throughout the larger project discussed here has not only been to study the shapes of vases, leaves and shells, but to showcase diffeomorphic shape analysis methods for studies on object shapes, from images, as a whole. We have seen how well diffeomorphic methods, particularly SRVF methods, have performed across the board, whether it's the Karcher mean computations compared to the standard means, or the classification results when paired with a k-NN classifier. Diffeomorphic shape analysis methods are seldom applied to real-world applications, but this project has shown that they can be. Ultimately, we have shown that diffeomorphic shape analysis methods belong in every applied scientist's toolbox.

4.2 The Effectiveness of Images

In this modern era, analyses studying the shapes (or forms) of objects is very often carried out via images of the objects. There are many advantages of using images; from the speed at which an outline can be measured, to the accessibility of the object in the first place. However, literature regarding the *effectiveness* of utilizing images in the analysis of shapes, in comparison to other techniques, is largely absent. This was the motivation to take on a project to delve into the question.

Traditionally, the shapes of objects were measured by hand. Today, with the advancements in technology, 3D laser scanning is slowly becoming a popular alternative (see for example, the geometric morphometrics analysis on 3D-scanned vases in (Joháczi, 2018)). Both of these methods are important as they produce high-quality and accurate measurements of an object, which may or may not be similarly obtained from an image. By returning to the Greek vases, we compare the effects of these methods in classification.

4.2.1 Measuring the Shapes of Vases

The shapes of vases were often studied by measuring the outlines of the vase at various angles, by hand, as seen in (Bloesch, 1940). Though this method of computing outlines can be tedious, time consuming and require direct access to a vase, it is still used today, as it is (rightly) believed to be a highly accurate method in analysing shapes. For example, in (Mackay, 2010), measurements taken by hand, led to a study of the vase shapes in order to establish a chronology belonging to a notable potter, *Exekias*[104].

Modern-day laser scanners can also be used for *accurate* measurements. 3D laser scans can achieve high quality three dimensional scans of an object, Here, we obtain more information about a shape of an object, than any other method, whether hand-measured or extracted from an image using computer vision tools. However, this approach is rather slow[105] and requires expensive hardware.

There's no doubt that extracting a contour from an image of a vase is quicker and easier than measuring an outline by hand or with a laser scanner. However, there are possible

[104]https://www.britishmuseum.org/collection/term/BIOG58234

[105]can take between a couple of minutes to over an hour.

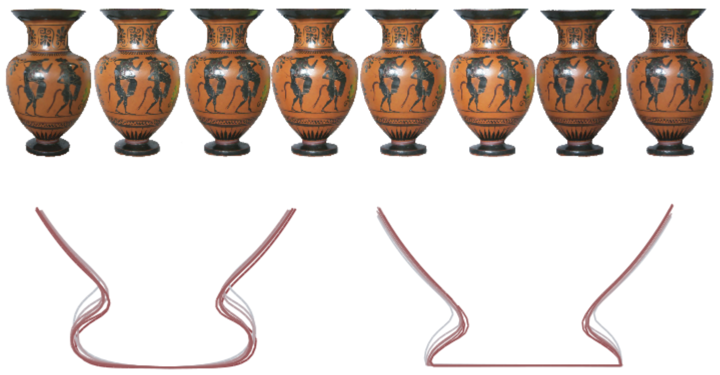


Figure 4.19: Distortion via perspective projection. Top: Photographs of an amphora (at Victoria University of Wellington), taken at varying distances from the vase. Bottom left: the base of the contours from the images above. Bottom right: modified version of the contours, in order to straighten the bases. This figure demonstrates how the straight line describing the base becomes curved due to perspective projection.

issues with this technique that can certainly arise. These can include / be caused by the following:

- Poor quality images that can render binarization and contour extraction algorithms useless. This issue is particularly likely to occur when handling old images, such as some that may be seen in the Beazley archive[106].
- Distortion caused by the camera lens, which causes straight lines to appear curved. An extreme example of this is caused by wide-angle lenses such as a Fisheye lens[107].
- Distortion by *perspective projection*. In other words, straight lines appearing curved, but instead of being caused by a certain type of lens, it is the result of the photograph being taken too close to the object. This phenomenon can be seen in Figure 4.19, where the same vase has been photographed from different distances, but from the same angle. Here, the plot on the right shows just how curved the base of the outline vase becomes, despite originally being straight.[108]

[106] https://www.beazley.ox.ac.uk/carc/pottery

[107] https://en.wikipedia.org/wiki/Fisheye_lens

[108] We note that to cater for such effects, a collaboration between a computer scientist and vase historian led to a program designed to estimate the degree of distortion, in (Wyvill and Anson, 2004). However, this program cannot *automatically* detect and fix distortions, so it cannot be easily used at a larger scale.

4.2.2 Project Overview

In order to evaluate the effects of employing images in the analysis of shapes, we designed a project based on the shape classification of ancient Greek vases. Here, we collated datasets of outline contours of Greek vases, obtained from hand-measurements, 3D laser scans, and images. Henceforth, we incorporated a classification approach (as seen in Section 4.1) on the three datasets, and compared the results.

Data Processing

We begin with three datasets of Greek vases:

- Hand-measured *amphorae* vases (containing side outlines from numerous angles of the vase), by Dr. Anne Mackay of University of Auckland,
- Profiles taken from 3D laser-scans of *lekythoi* vases, by Dr. Szilvia Joháczi, from Eötvös Loránd University, Budapest,
- 2D images of *amphorae* and *lekythoi* vases.

Although the *manually-measured* datasets (i.e., from hand-measurements, and 3D laser scans) are based on differing vases[109], the image-based dataset covers all the vases that appear in the other two. Furthermore, to obtain outlines from the image dataset, we incorporated Algorithm 6.

In order to create a dataset of smooth, symmetric outline contours of vases, that also take into consideration the possible distortions discussed in the previous section, we take on the following steps:

1. Split the contour into side contours[110],
2. Compute the Karcher mean (see Section 2.5.3) of the side contours,
3. Connect the Karcher mean with its reflection (along the y-axis) in order to create a symmetric vase.

[109]Ideally, we would have both datasets based on the same vases, but unfortunately this could not be obtained within the scope of the project.

[110]Note that for the hand-measured vases, we already have numerous side contours. For the other two datasets, this step implies splitting an outline contour in half, through the vertical axis.

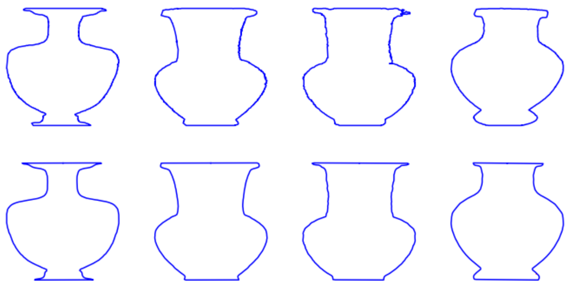


Figure 4.20: Before and after. Top: contours obtained from images via Algorithm 6. Bottom: Symmetrized contours after the 6-step contour procedure.

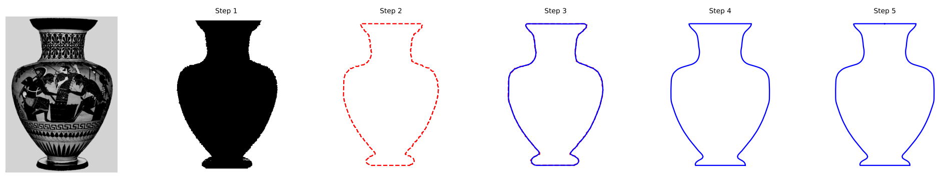


Figure 4.21: An example of an amphora (Beazley number 9029664) going through the various data processing steps. Here, we begin with the steps outlined in Algorithm 6, which involve (i) Image binarization, (ii) contour extraction, and (iii) contour smoothing. Following on from this, the 6 processing steps are taken to obtain a symmetric vase. Note that the Step 3 image shows both the original outline contour (dotted red) and the smoothed contour (blue). On average, in total, it takes approximately 4 seconds to complete for a single image, with the bulk of that time being spent on symmetrisation[111]This process time assumes manual modifications to the image, if needed; for example, handle-removal via image-editing software has already been completed.

4. **For image outlines only:** Modify the base and top of the vase, by creating a straight cut, to counter the possibility of distortion by perspective projection (as seen in Figure 4.19).

5. Procrustes alignment between new outline and a template.

6. Reparametrize outlines to have 250 equally arc-length spaced points.

An example of the specific steps taken for a vase image in particular, is shown in Figure 4.21, whilst further examples of original contours and *final* contours can be seen in Figure 4.20. Henceforth, we obtain three datasets of symmetric outline contours from Ancient Greek vases, based on hand-measurements, 3D scans, and images.

[111]On a machine with an Intel Core i7-8550U @ 1.80GHz CPU and 8GB of RAM.

Classification

Recall that our aim is to compare the performances of image-derived contours in classification, with alternative, more *accurate* methods. For this reason, and due to the fact that the two non-image datasets were originally of differing contours (amphorae and lekythoi), we group the manually-derived contour datasets (i.e., from hand-measurements and 3D scanned vases) together. Furthermore, based on the *success* of our previous classification tests on Ancient Greek vases (see Figure 4.18 for example), we incorporate a similar approach by utilizing a k-NN boostrapping technique (Algorithm 10) on distance matrices containing the distances between the pairs of vase contours using the SRVF Path-Straightening method (see Section 2.5.1), for each of the two datasets.

Figure 4.22 shows an example of the k-NN classification results. In the above plot, we see confusion matrix resulting from the distance matrix computed with the manually-derived contours, whilst in the bottom plot the distance matrix from the image-based contours is used. We find that both methods (of deriving contours) led to impressive classification results, despite similar mistakes being made across both sets. Furthermore, the F_1-scores from the manually-derived contours was slightly higher, with $F_1 = 0.96$, compared to $F_1 = 0.92$ for the image-based results.

Conclusion

Our findings show that while the hand-measured / 3D-laser scanned outlines did improve the overall classification results, as emphasised by the average F_1-scores we just saw, the improvements were modest.

In summary, we have seen impressive classification results obtained with a distance matrix based on image-extracted contours, that can certainly compare to the classification results obtained via the other two vase-outline methods. Therefore, given this performance, and the fact that contours are far more easily obtained via images than other methods such as hand-measurements and 3D laser scans, we conclude that images are indeed an effective and arguably the most worthwhile method of shape classification.

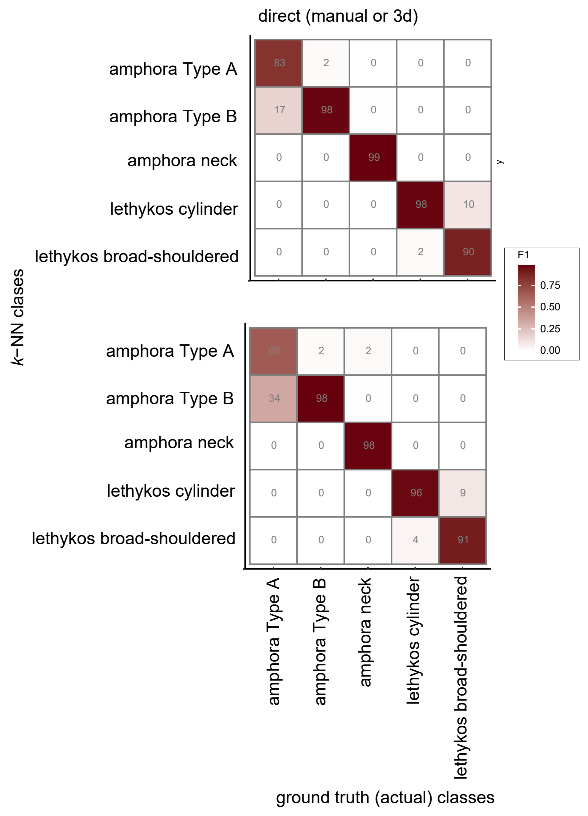


Figure 4.22: Top confusion matrices from 100 k-NN boostrapping tests. The top plot uses the distance matrix based on the manually-derived vase contours (from hand-measurements and 3D-laser-scans), whilst the lower plot uses the distance matrix obtained from the image-extracted outlines. This plot emphasises the similarity in the results between the three methods of measuring vase outlines. Furthermore, both methods performed well in classifying vases, echoing the results from Section 4.1.

4.3 Classifying Mussels

There are countless ways in which image classification is a highly practical tool in the everyday world; whether it is to identify anomalies in MRI scans (Abdullah et al., 2011), or provide facial recognition systems (Julina and Sharmila, 2017). In this project, we consider an example of: mussels.

4.3.1 Objectives

Bivalve[112] molluscs are creatures that often *hook* themselves onto the hulls of boats. Though this can be problematic at times, for example by limiting the speed of boats, or bringing in invasive species, (Minchin et al., 2003), it could also present the opportunity to determine previous locations of boats, based on the species of mussels seen attached.

One practical application of image-based classification is based on the detection of invasive mussel species. Henceforth, in collaboration with marine biologists at Victoria University of Wellington, we explore the potentials of classifying mussels from images, in order to identify them as native species, or non-native and potentially invasive species.

4.3.2 Previous Classification Methods

Classification of mussels from images has been done before, such as in (Magbayao et al., 2020), where classification was performed on images of Asian Green Mussel (*Perna Viridis*) to identify sex. In this particular example, the colour of the mussel gonad was of sole interest; where image processing was applied to obtain RGB values of the gonad, followed by classification on the RGB values using Fuzzy Logic[113] and a traditional k-NN classifier. However, here we are interested in the images of the shells.

An alternative approach is one attempted by our collaborators at Victoria University of

[112]The term *bivalves* refers to molluscs whose shells can be considered as consisting of two sides, hinged together, such as the shells seen in oysters, scallops, and mussels.

[113]Broadly speaking, fuzzy logic can be thought of as traditional logic, but with an added *vagueness* to it. Here, truth values are not restricted to just 0 and 1 and can instead inhabit any $p \in \mathbb{R}$, $0 \leq p \leq 1$. Multiple values can be input into one variable (where in the example above, the three RGB values were used), whilst the output of fuzzy logic, is one value. A condition is then used used to *classify* the objects, based on that one value. For more information, we refer the reader to (Zadeh, 1996).

Sub-species	# Samples
NatMg	70
MedMg	57
MlMa	53
OsMa	59

Table 4.2: Total number of mussel images per sub-species, in our dataset.

Wellington, with the SHAPE[114] program. The software uses Elliptical Fourier Descriptors to extract outline contours from images, (Iwata and Ukai, 2002). This can be followed by Eigenshape Analysis (see Section 4.1.7) on the Fourier coefficients, which can then be used to classify the mussel contours. We note that though this software can work well on finding precise outlines[115], the contours are not part of the output which the user receives, thus it cannot be used in conjunction with any other methods or classifiers.

4.3.3 Method

Owing to the successful performance of diffeomorphic methods on shell classification in Section 4.1 (see for example Figure 4.18), here, we adopt a similar approach. Specifically, as the top two methods employed an SRVF Path-Straightening algorithm, and Geometric Currents, we focus on the incorporation of these two methods in k-NN classification.

Dataset

In order to test whether an elastic shape analysis approach would perform well in classifying mussels from their outlines, we collated a sample of 239 contours from images of native and non-native mussels, provided by our collaborators. This dataset consists of mussels from two species: *Mytilus galloprovincialis* and *Mytilus aoteanus* that were split into two subspecies, based on the location that they originate from. The non-native *Mytilus galloprovincialis* species comprised of mussels from the North Atlantic and the Mediterranean seas; that are subsequently labelled as *NatMg* and *MedMg* respectively. The *Mytilus aoteanus* species are native to mainland New Zealand and its off-shore islands, and are labelled *MlMa* and *OsMa*. The total samples can be seen in Table 4.2.

[114]http://lbm.ab.a.u-tokyo.ac.jp/~iwata/shape/

[115]albeit with the requirement of manual parameter selection which cannot be automated.

Mussel Image Processing

As always, the first step in image-based classification is image processing. In order to obtain mussel contours, we incorporate Algorithm 6, as detailed in Chapter 3. This algorithm begins with a binarization step, based on Algorithm 5, that aims to segment an object centred in an image, whilst *filling-in* the interior of the object. Recall that this algorithm works by computing a threshold value based on the mean average background colour, and the computation of two ellipses. While the larger ellipse is used to segment the object, and the smaller ellipse is used to *fill-in* the interiors, the values inbetween the two ellipses go through a *nearest-neighbour* inspired approach, based on the threshold value, to determine the colours of the pixels within the boundary. Once binarized, Algorithm 6 employs the marching squares algorithm (Lorensen and Cline, 1987) to find the outline contour of the mussels in the images, followed by a snakes algorithm (Kass et al., 1988) to smooth the contour. Using linear interpolation, contours were reparametrized to have 251 points. Recall that for our project on Gastropod shells (in Section 4.1.11), we based our contours on 150 points across the outline boundary of the shells, as this number was sufficient enough to maintain the necessary shell shape information, whilst minimizing computational costs. The differentiation between the shapes of mussel shells across the species is far more subtle, thus a greater number is required. Henceforth, through impromptu evaluation[116], we found that 251 points were needed.

Elastic Shape Analysis & Machine Learning

In order to perform classification on the shapes of mussels, we first compute two distance matrices, based on the pairwise distances between mussel contours using the SRVF Path-Straightening algorithm and the Geometric Currents approach (see Sections 2.5.1 and 2.3.2 respectively). Furthermore, we return to our implementation of a k-NN classifier, as described in Algorithm 10. Recall that this implementation involves *bootstrapping* the classes, where training and testing sets[117] are randomly generated N times, from our dataset, and the F_1-scores are obtained at each of the N iterations. Here, we set $N = 50$ and evaluate the average F_1-score using the two distance matrices.

We plot an overview of the scores in Table 4.3. At first, the F_1-scores resulting from both

[116]We note that such evaluations were based on kNN implementations on distance matrices obtained from contours with a varying number of points, ranging from 150 to 750.

[117]In this project, the training and testing sets each contained 50% of the total samples.

# Classes / Method	σ	Median	Mean	Max
4 / SRVF	0.033	0.532	0.529	0.622
4 / GC	0.038	0.550	0.551	0.621
3 / SRVF	0.031	0.698	0.697	0.761
3 / GC	0.029	0.715	0.712	0.789
2 / SRVF	0.028	0.750	0.752	0.800
2 / GC	0.023	0.774	0.773	0.825

Table 4.3: Average F_1-scores (where σ represents the standard deviation) from k-NN classification using the distance matrices obtained from SRVF Path-Straightening (*SRVF*) and Geometric Currents (*GC*) on mussel contours, with varying number of classes, based on certain combinations of sub-species. The top rows display the average F_1-scores, where the classes were kept as the original four sub-species. Meanwhile, in the middle rows, the two non-native sub-species have been combined, whilst the native sub-species remain separated. Finally, the last rows show the average F_1-scores after the four sub-species have been grouped into two classes, representing all native mussels, and all non-native mussels.

distance matrices were rather poor, averaging around the 0.5 mark. However, as our global aim is to differentiate the non-native species from the native species, we decided to combine the subspecies together, to see how this could affect the results. We found that when the non-native species are combined together, the F_1-scores drastically improve, as can be seen in the middle rows of Table 4.3. These results are particularly impressive when the Geometric Currents method is employed. Similar improvements resulted when the two native species were also combined and thus classification is performed on only two classes. Here, we find a top score of 0.825, resulting from the Geometric Currents method.

Thus far, we have achieved decent results, given the similarities of the shapes of the four mussel sub-species. This similarity is emphasised in Figure 4.23, which shows the Karcher means of each of the four classes. Considering the within-species likeness between the mussels, it is therefore no surprise that the F_1-scores improve when the classification is performed on species, rather than sub-species.

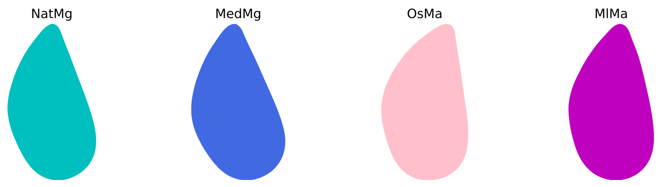


Figure 4.23: Karcher means computed on the four classes of mussels. These plots emphasise the within-species similarities of the shapes of the mussels, particularly within the *Mytilus galloprovincialis* mussels (cyan/blue). Though we note that differences between *Mytilus galloprovincialis* and *Modiolus aoteanus* (pink/magenta) are visible to the eye, especially in the Karcher mean computed from the off-shore islands samples (*OsMa*, pink).

4.3.4 Summary

In comparison with classification scores our collaborators obtained in previous tests, the top F_1-scores here were reasonable. However, this was only after classes were combined. This suggests that shape methods may not be enough for the classification of mussels from images, and thus we may need to incorporate additional features in future work.

Overall, our results reveal two important consequences. Firstly, we find that the Geometric Currents method significantly outperforms the SRVF Path-Straightening method despite the fact that the SRVF Path-Straightening method constantly came out on top in previous tests, seen in Section 4.1. This surprising outcome hints that Geometric Currents should be considered as one of the top methods for use in shape classification, particularly, when the shape differences in the dataset are quite subtle. Moreover, as Geometric Currents has a much faster processing time than SRVF Path-Straightening, these results motivate the need for further tests, allowing us to take an even deeper look into comparing these two methods. Finally, the second consequence of our findings, is that despite the imperfect scores, we *can* certainly classify mussels by their shapes and obtain useful classification results. Our work here encourages us to continue developing methods with our collaborators, with a next step of expanding our training sets, in order to create a tool that automatically and successfully distinguishes non-native mussels from native mussels, by analysing their shapes.

Chapter 5

Applications of Elastic Shape Analysis on Open Curves

In Chapter 4, we presented an anthology of projects based on the classification of closed shapes extracted from images. In this chapter, we turn our focus onto open curves, found in non-image data. We use elastic shape analysis for two practical applications in wildlife conservation.

1. **Kākāpō Health Classification** – We focus on a collaborative project with conservation biologists at Kākāpō Recovery[118], which looks at incorporating elastic shape analyis to build a tool to automatically classify the health status of kākāpō chicks.

2. **Kiwi Call Identification** – In this project, we work with audio data containing recordings of kiwi calls. We design an algorithm to extract curves from spectrograms of audio files, and combine methods of elastic shape analysis with machine learning to develop a procedure for identifying individual kiwis, based on the shapes of their calls.

[118]https://www.doc.govt.nz/our-work/kakapo-recovery/

5.1 Kākāpō Health Classification

Flightless, nocturnal, endemic to New Zealand, and the heaviest parrot in the world. Internationally adored, two-time winner of national Bird of the Year[119] competition, and one of them[120] was even appointed as the official Spokesbird for Conservation. Despite the love and fame, kākāpō (*Strigops habroptilus*) are critically endangered, with only 201 alive today. Thanks to the efforts of the Kākāpō Recovery[122] team (part of New Zealand's Department of Conservation), this number is a significant increase from the mere 50 or so that were alive in the 1990s. Therefore we might ask ourselves: *what can we do to help?*

Like all natural beings, kākāpō, and particularly, kākāpō chicks can grow at different rates; some may be slower than others, whilst some might reach their peak in half the time. This is where elastic shape analysis has the potential to play a useful role in analysing the growth curves of this endangered species. This was the motivation to start a project, in collaboration with Kākāpō Recovery, to examine whether tools from elastic shape analysis can aid in the monitoring, and hence the conservation, of these marvellous creatures, by analysing the shapes of their growth curves.

5.1.1 Growth Curves

Growth curves are a useful tool in analysing biological data, whether it's measuring the population of a certain bacteria, or the differences in skull size of polar bears, (Bechshøft et al., 2008). When it comes to the conservation of avian species, age-vs-weight growth curves are increasingly helpful for various reasons from the relative ease of taking weight measurements (in comparison to other types of measurements), to the relationship that weight can have on further variables, such as breeding (see (Elliott et al., 2001)). Growth curve analysis is of particular importance in the early stages of life. This is because in many species, the prominent changes in growth occur early on in their lives, and can additionally prove pivotal in the existence of the individual into adulthood. Therefore, for our project, we focus our interest on the growth curves of chicks.

[119] `https://www.birdoftheyear.org.nz/`

[120] In 2010, Sirocco[121] the kākāpō, an international superstar, was appointed as the official Spokesbird for Conservation by the then New Zealand prime minister.

[121] `https://www.doc.govt.nz/nature/native-animals/birds/birds-a-z/kakapo/sirocco/`

[122] `https://www.doc.govt.nz/our-work/kakapo-recovery/`

Common models for growth analysis on avian species include Gompertz and Von Bertalanffy models. In particular, these were seen in research done on the weights of avian species such as blue tits & great tits, and the southern & northern ground hornbill, in (Brunner et al., 2021) and (Engelbrecht et al., 2007) respectively. The models can be written in a variety of ways, including the following:

Gompertz:

$$G(t) = W_0 \exp\left(\ln\left(\frac{W_A}{W_0}\right)(1 - e^{-tg})\right) \tag{5.1}$$

Von Bertalanffy:

$$B(t) = W_A(1 - \exp(-(t - t_0)g)^3 \tag{5.2}$$

where W_0 is the initial weight, W_A is the asymptotic weight (the weight that the growth curves tend towards); g is the average growth rate or growth coefficient, and t_0 is a known theoretical age where the weight is zero. Note that the power of three in Equation (5.2) is not always seen in Von Bertalanffy functions, for example with length data. However, it is often included when analysing weight data.

Finding suitable parameters for the above growth models is not always a simple task. Often, a minimizer, such as a least squares minimizer, is employed to optimize the parameters. To test this, we incorporated kākāpō weight data, and used a least squares algorithm to fit the Gompertz and Von Bertalanffy curves to the weights of 61 chicks aged 0-90 days. The results can be seen in Figure 5.1. Here, we modelled the curves using two weight means: a Euclidean mean, and a Karcher mean. Interestingly, the two models appear very similar when their parameters are optimized. Another remark is that though the Euclidean mean is naturally unsmooth, a growth model such as Gompertz or Von Bertalanffy can be used to smooth the mean. This can be useful in some cases, for example, if we required a smooth template for a mean, to be used in further classificaiton. However, we note that there's also the risk that we may miss information from the mean curves, by such smoothing. For more details on the optimization of these and similar models, we refer the reader to (Kühleitner et al., 2019).

The incorporation of a Karcher mean is a unique twist in the analysis of these growth models, that traditionally rely on standard Euclidean means. This is a basic example

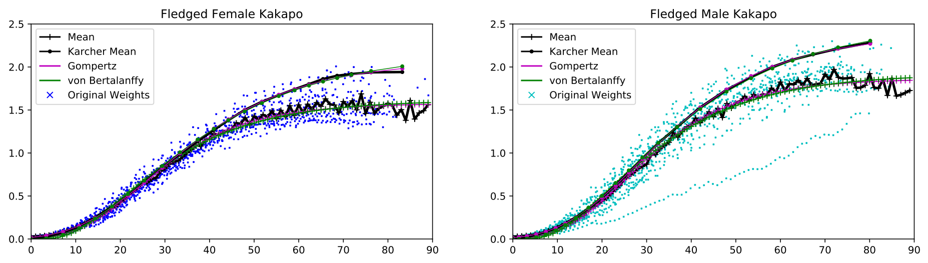


Figure 5.1: A Gompertz and Von Bertalanffy curve fitted to kākāpō chick weights, aged 0-90 days, for male (left) and female (right) kākāpō chicks. The Gompertz and Von Bertalanffy functions are fitted to the Karcher mean (see section 2.5.3) of the growth curves (black, circled), and to the standard Euclidean mean of growth curves (black, crossed). These plots show us that the Karcher mean of kākāpō weights is much smoother than the standard mean. Moreover, we see that both the Gompertz and Von Bertalanffy curves fit almost perfectly to the Karcher mean. This suggests the potential that the Gompertz and Von Bertalanffy functions may have in the analysis of kākāpō growth curves, using elastic shape analysis. We remark that the Karcher mean curves in both plots are much higher than the standard means. This is due to the scaling-invariant metric used in the Karcher mean, which meant that the curves had to be *re-scaled* in order to be plotted on top of the original curves, causing a possible distortion in the y-axis. Though this does not matter when analysing the *shapes* of the curves, it could be a concern if a direct comparison is being done with the original y-values. Note that we will come back to the point later on in this section.

of combining elastic shape analysis with traditional growth modelling methods. The advantage of using elastic shape analysis is that we can focus on the *shape* of the growth curves. This is particularly helpful when comparing multiple growth curves, as the time domain is, in essence, *ignored*, enabling more meaningful comparisons to be made between the growth of multiple chicks, even if they grew at different rates. Although we did not pursue this particular path of incorporating growth models with elastic shape analysis much further, further comparisons between Karcher means and Euclidean means in growth curve modelling might prove to be an interesting research avenue to explore in the future.

5.1.2 Kākāpō Growth Monitoring with Shape Analysis

Monitoring the weights of kākāpō regularly is a vital step for their conservation. From the day they hatch, they are weighed periodically, particularly in the weeks before they

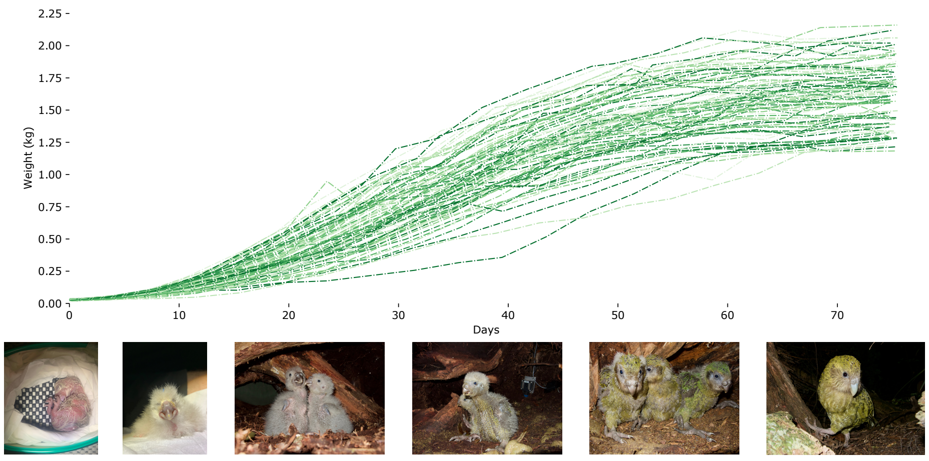


Figure 5.2: Growth curves of a sample of chicks aged between 0-75 days, alongside images of growing chicks, from birth to fledged. Images courtesy of Dr Andrew Digby from Kākāpō Recovery[124]. The first photo was taken on the day a chick has hatched, the second-to-last photograph is of *almost*-fledged chicks, and the last photo shows a fully fledged chick.

fledge[123]. Kākāpō chicks grow a lot during this period, as can be seen in Figure 5.2, hence the importance of regular measurements. For this reason, the Kākāpō Recovery team actively monitor these weight curves, and use them as the main method of detecting poor health conditions of chicks.

Currently, to monitor the growth of chicks, the team compute the Euclidean mean weight of all the male or female chicks, and a threshold is set at the minus 20% mark[125]. Chicks that fall below this threshold are classed as *underweight* and provided with extra help.

In the subsequent sections, we detail a new tool we have created in collaboration with the Kākāpō Recovery team, which uses elastic shape analysis to monitor kākāpō. Though the focus is on analysing kākāpō chicks from the day they hatch, the new platform will also work to analyse the health of older, post-fledged chicks too. This tool has one simple aim: to classify a chick as *healthy* or *unhealthy* by analysing their growth curve.

[123]In these flightless birds, fledging is considered to be when the chick leaves the nest and learns to fend for and feed themselves. This usually happens at around age 75-120 days but can vary hugely, particularly for hand-reared birds.

[124]`https://www.doc.govt.nz/our-work/kakapo-recovery/`

[125]Specifically, two thresholds are set, where one is at the 20% mark, and the other at 35%, where the latter is a cause for a greater concern. For succinctness, in this section, we will focus on the 20% mark.

5.1.3 Kākāpō Weight Data

As technology advances across the world, so do the gadgets used for kākāpō conservation. Each bird has a smart transmitter, all nests are fitted with infrared cameras during breeding season, there are data loggers scattered in various locations, automatic scales, and more[126]. This is relevant to our project, as it not only enables us to gather up-to-date weight data, but it provides us with additional information, such as the location, which we could possibly use in our algorithms.

Each kākāpō is weighed regularly. When chicks first hatch, they are weighed manually. Meanwhile, the adult birds can be weighed when they visit a feeding station, and their weight is automatically added to the database with a hidden data logger; in addition to being weighed manually from time to time. All weights, and other information such as locations, and behavioural details (e.g. mating) are logged onto a database, which can be accessed online.

The Rimu Effect

For some time now, studies have indicated that kākāpō breed in response to the masting of certain trees, and in particular, (due to the islands that most kākāpō reside in) the Rimu tree (*Dacrydium cupressinum*), (Fidler et al., 2008).

Rimu are large, slow-growing, coniferous trees that mast (i.e. bear lots of fruit) depending on their environment's temperature approximately every 2-3 years. Even if the fruit grow, if the climate is not warm enough, the fruit will not ripen.

Though there still remains secrets to be discovered, regarding the breeding of kākāpō, it is accepted that kākāpō breed in response to the rimu mast, (Harper et al., 2006). However, for chick growth, it is not the *abundance* of fruit that is relevant to the kākāpō per se, but the proportion of ripe fruit. Henceforth, the ripe rimu *status* of the year a chick was born can have an important impact on a chick's weight, and consequently, their development, (Cottam et al., 2006).

[126]For more details on the technologies used, check out the Kākāpō Recovery[127]website.
[127]`https://www.doc.govt.nz/our-work/kakapo-recovery/what-we-do/technology/`

Data Categories

In order to analyse growth curves, we require pairs of *(weight,date-time)* data for the chicks, as well as the hatch date to compute age. We incorporate additional factors which can affect growth, such as the ripe Rimu effect which we discussed previously. During our tests (some of which will be shown in the subsequent sections), we found that three factors had a distinctive effect on growth curves:

1. **Sex**: Chicks are of similar sizes when born, and identifying the sex can take days and sometimes weeks. However, differences in growth curves soon appear, and classification results tend to improve when a chick is identified and grouped with its sex as soon as possible. Thus we categorise chicks by their sex.

2. **Ripe Rimu**: As the *ripe rimu status* of the year a chick is born can affect the development of a chick, we include this factor in our dataset. We categorise each chick in our sample by introducing a binary ripe-rimu variable, depending on the *ripe rimu status* of the location in the year where the chicks were born.

3. **Hand-Reared**: There are times when chicks might need to be pulled for hand-rearing. This could be for various reasons, for example, if they are not doing well, to encourage mothers to breed again, or if the chick is unwell and requires more medical attention. Hand-rearing chicks is complicated, but there are certainly great benefits to it, especially when it comes to saving the lives of kākāpō that may not have survived otherwise. On the other hand, hand-rearing chicks can also lead to slower growth, as discussed in (Eason and Moorhouse, 2006). For this reason, in this project, we also categorise the chicks in our sample, based on whether they have been hand-reared or not. As chicks can go in and out of a hand-rearing facility numerous times, to class a chick as being *hand-reared*, a boundary[128] was chosen by our collaborators. Thus, if a chick had been hand-reared for more than the boundary, from age 0 - 75 (days), they were classed as *hand-reared*.

Training Data

Our aim is to create a tool that can detect whether a chick is *healthy* or *unhealthy*, based on their growth curve. To train an algorithm in the process of building our health detec-

[128]During our tests, a chick's hand-reared status was set to "Y" (*yes*) if it had been hand-reared for 30 or more days, and "N" if it was hand-reared for less than 10 days. Those in-between had the option of being classed somewhere in between as "YN", as "Y" / "N", or just excluded from the training sets.

tion tool, we require some sort of *ground truth* classification for the training data.

We based our *ground truth* definition of *unhealthy* on two things:

1. Whether a chick had survived to a *fledged* age (approximated here as 75 days).

2. Whether the chick appeared on a list of chicks that were observed to be *unhealthy* or underweight, collated by the team at Kākāpō Recovery, including their vet.

Overall, our dataset of chicks contains weight information for each chick over time, as well as *meta* details, i.e. sex, date-hatched, and the hand-reared and ripe-rimu statuses. Our training set also includes the binary true health classification.

5.1.4 Karcher Means of Kākāpō Growth Curves

In order to monitor kākāpō chicks, conservationists routinely monitor their weights by comparing it to the standard ($\mathbb{L}^2$) mean. In order to compute this average, which is often associated with the moniker *cross-sectional sample mean*, we do the following:

$$\bar{\mu} = \arg\min_{\mu} \sum_{i}^{N} ||f_i - \mu||_{\mathbb{L}^2}^2 \tag{5.3}$$

where $f_1, \cdots, f_N$ represent the growth curves.

Like all natural beings, kākāpō chicks can grow at different rates; some may be slower than others, whilst some might reach their peak in half the time. Therefore, comparing chicks to a standard mean might not be the most credible method to use, and this is where tools from shape analysis, such as the Karcher mean, could be favourable. As seen in Section 2.5.3, a Karcher mean is defined with a chosen distance metric, d, based on a geodesic equation defined on the shape space.

As seen in the background material chapter, we can define a distance metric, which involves an alignment that is invariant to shape-preserving transformations. This metric (d) can then be used to compute a mean:

$$\bar{\mu}_{\text{KM}} = \arg_{\mu} \min \sum_{i}^{N} d(f_i, \mu)^2 \tag{5.4}$$

The two mean equations seen in Equations (5.3) and (5.4) are equivalent if the distance metric in Equation (5.3) is simply $\mathbf{d}(f_i, \mu) = f_i - \mu$. This is the standard arithmetic mean:

$$\frac{d}{d\mu}\sum_i^N (f_i - \mu)^2 = 0 \implies -2\sum_i^N f_i + 2N\mu = 0 \implies \mu = \frac{1}{N}\sum_i^N f_i.$$

In order to analyse our growth curves, in this project, we will choose a distance metric that focuses on an alignment between the *shapes* of the growth curves, as seen in the Equation (2.28) of Section 2.4.4. We will use this to compute Karcher means, as given in Algorithm 3 in the Background chapter. We will incorporate the Karcher mean in further kākāpō growth analysis, and will compare the results to those obtained via the use of the standard mean.

Some Karcher means can be seen in Figure 5.3. The first thing to note is that these have been plotted alongside the original sampled growth curves, and not the aligned growth curves. Though this has been done for graphical purposes, to show both the growth curves and the Karcher means in one plot, it is a useful reminder that Karcher means may alter the original scaling and positioning of the growth curves in their computation, due to the invariances imposed on its distance metric. Recall from Chapter 2, that we consider shapes as curves modulo shape-preserving transformations. Thus, the metric we describe in the Karcher Mean algorithm (Algorithm 3) in Section 2.5.3, is invariant to shape-preserving transformations. Henceforth, due to this invariance to scaling and translation in particular, the resulting domain and range of the Karcher mean, will differ from the original curves. For this reason, when plotting a Karcher mean alongside the original growth curves, as done in Figure 5.3 and in Figure 5.1 for example, the Karcher means would need to be translated and re-scaled[129]. Therefore, when studying such plots, we must not dwell on the values presented on the y-axis.

Figure 5.3 shows us a few interesting things. Firstly, we can see that there are clear differences between the growth curves of chicks in the relevant groups. For example, in the top graphs, the growth curves for hand-reared chicks are much lower in weight than those chicks who weren't hand-reared; and this is true for both male and female chicks. Secondly, we can see that there are distinctive differences in the Karcher means too: for

[129]Note that this scaling is usually based on the average ranges.

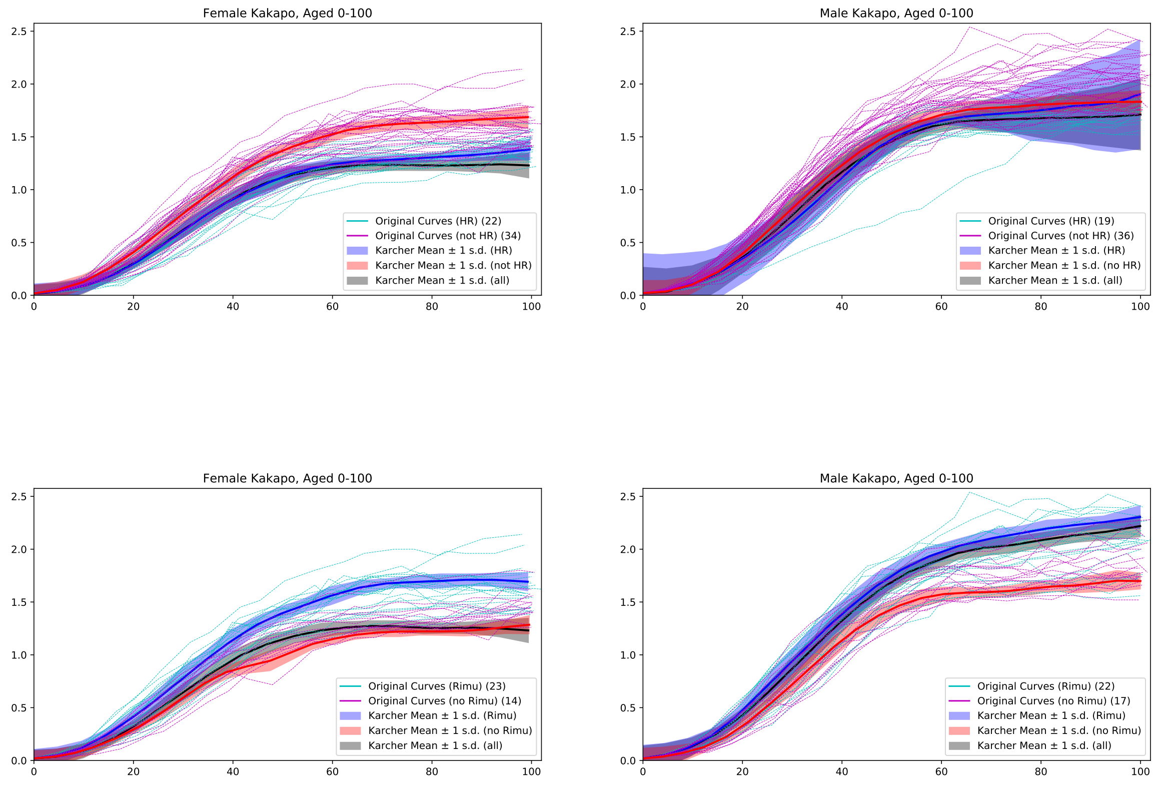


Figure 5.3: Karcher means of chicks aged 0-100 days. Top: Karcher mean from growth curves of hand-reared (cyan) vs non hand-reared (magenta) chicks. Bottom: Karcher means from ripe-rimu (cyan) vs non ripe-rimu (magenta) chicks. Sampled growth curves are also plotted (dotted, cyan or magenta). An additional Karcher mean (black) is computed on the total set (i.e. without separating the samples based on their hand-reared or ripe-rimu status). Note that the Karcher means here are plotted alongside a boundary created by ± 1 standard deviation, computed point-wise across the aligned samples. These graphs show us that differences can certainly be seen in the means when separating the chicks based on their hand-reared or ripe-rimu status. The plots also show that Karcher means on the full set generally resembles the Karcher mean of one set more than the other - this could potentially be due to the template curve that initially chosen.

example, in the bottom graphs, the Karcher mean of the chicks born during a ripe Rimu season, *stabilizes* more gradually than the Karcher mean of the non-Rimu chicks. The other curious point the graphs show is that when computing the Karcher means of the groups combined, the mean tends to heavily resemble one of the Karcher means of the divided groups. These points all indicate that splitting the chicks by the factors of sex, ripe-Rimu, and hand-reared could be helpful, and are therefore appropriate to measure in the data.

5.1.5 Distances between Growth Curves & Means

Our aim is to classify or sample chicks into *healthy* and *unhealthy*. Often in the literature, this may involve a distance computation between a sample point and the mean, such as in machine learning with a K-means clustering algorithm. In order to investigate distance computations on our kākāpō growth curves, we tested out two methods, both based on a distance metric within the SRV framework, as seen in Section 2.4.2.

In Chapter 2, we saw that for two open curves, $c_1, c_2 : I \mapsto \mathbb{R}^2$, on some interval, I, the square root velocity functions (SRVFs) can be described by the mapping:

$$\mathbf{q}(c(t)) = \frac{\dot{c}(t)}{\sqrt{|\dot{c}(t)|}}. \tag{5.5}$$

In Section 2.4.2, we saw that the space of all such SRVFs is analogous to the space of square-integrable functions, $\mathbb{L}^2(I, \mathbb{R}^2)$. Hence, we can employ the $\mathbb{L}^2$-distance in this space, and we define an SRVF distance, $\mathbf{d}$:

$$\mathbf{d}(c_1, c_2) = ||\mathbf{q}(c_1) - \mathbf{q}(c_2)||_{\mathbb{L}^2}, \tag{5.6}$$

for two open curves, c_1, c_2.

We use this to outline possible distance methods for use in this project:

Distance Method 1: Compute the geodesic distance directly between a growth curve in the testing set to the Karcher mean computed from a training set. This method registers the growth curve to the Karcher mean using the pairwise registration method described in the Background Material chapter. The resulting distance, d, is the SRVF-distance between the registered growth curve and the Karcher mean:

$$d = \min_{\gamma} \mathbf{d}((q_{gc}, \gamma), q_{KM}) = \min_{\gamma} \left(||(q_{gc}, \gamma) - q_{KM}||_{\mathbb{L}^2}\right) \tag{5.7}$$

$(\cdot, \gamma)$ is the action by composition (as seen in Equation (2.22)), and q_{gc}, q_{KM} are the square root velocity functions of the testing growth curve and Karcher mean respectively.

Distance Method 2: Using the resulting γ-functions (registration functions) from the Karcher mean computation on the training set, we calculate the average, $\bar{\gamma}$. We then reparametrize a growth curve by $\bar{\gamma}$ and compute the SRVF-distance between itself and the Karcher mean.

$$\gamma_i = \arg\min_{\gamma} ||(q_i, \gamma) - q_{KM}||_{\mathbb{L}^2} \tag{5.8}$$

$$\bar{\gamma} = \frac{1}{N}\sum_{i}^{N} \gamma_i \tag{5.9}$$

$$d = ||(q_{gc}, \bar{\gamma}) - q_{KM}||_{\mathbb{L}^2} \tag{5.10}$$

where the q_i are the square root velocity functions of the growth curves in the training set, and q_{gc} is a growth curve from the testing set.

In order to classify a growth curve, we can compare the distance computed to a chosen threshold, as we will see later on in this section. This threshold value can be optimized in various ways, for example by utilizing a cross-validation grid-search approach.

Distance Method 1 provides us with a more accurate approximation of how close a growth curve is to the Karcher mean. However, Distance Method 2 removes the requirement of numerous optimization calculations, which noticeably speeds up the process. Given that our research is not theoretical, and is directly involved with these real-world applications, the significance of minimizing computational time mustn't be underestimated.

5.1.6 Boundaries on Growth Curves

An alternative to computing distances in the health classification of growth curves could be to incorporate a variation of the current method used by the Kākāpō Recovery group. Presently, a standard mean is computed for the growth curves, and a boundary is set at the mean$-$20% mark[130], as seen in the left plot in Figure 5.4. A chick whose curve falls below this boundary is then classified as underweight, hence, *unhealthy*.

Conversely, we can create a boundary using the Karcher mean. Unlike the standard mean, deviation both above and below the boundary are equally worrying, hence why we

[130]We reiterate that this 20% figure has been tried and tested by the experts at Kākāpō Recovery; as well as a separate, slightly more extreme, 35% threshold which is also sometimes used.

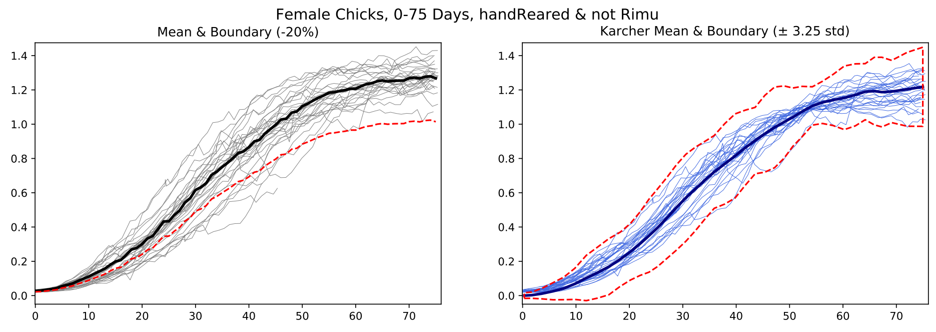


Figure 5.4: Example of growth curves of a sample of female chicks, aged 0-75 days. In the left plot, we see the mean (thick, black) alongside the original growth curves (thin, black). In the right plot, we see the Karcher mean (thick, blue), alongside the aligned growth curves (thin, blue). In both plots, the red dotted lines represent the boundaries. Note that the left plot does not have an upper bound as at this point it does not matter if a chick is *over-weight*. In the right plot however, as we are plotting a Karcher mean alongside aligned growth curves, if a curve appears above the Karcher mean, due to the invariances we discussed earlier, this can not necessarily be associated with a chick being *over-weight*.

don't stick to a single lower bound. Moreover, instead of decreasing by some percentage, a boundary can be created using the standard deviation of the Karcher Mean, as seen on the right, in Figure 5.4. Here, the boundary region is: Karcher mean $\pm 3.25\times$ standard deviation[131]. By using a standard deviation, we distribute significance at various ages, in a way that is more relevant to the data. A chick whose aligned growth curve falls outside of the boundary region will be classified as *unhealthy*; which is not too dissimilar from the previous method.

5.1.7 Method Comparison Experiment

Thus far, we have mentioned four possible methods which can be used to classify growth curves (two distance methods and two boundary methods). However, we have yet to make any substantial comparisons between these. In this subsection, we outline an experiment designed to test all four different methods of kākāpō growth curve classification.

Method 1: This is the original method, which involves creating a lower bound based on the standard (*cross-sectional*) mean minus 20%.

[131]We note that the standard deviation is computed from the aligned growth curves. The choice of value 3.25 is for this plot. Later on, we will see how this value is chosen in our algorithms.

Method 2: A variation of the original method, which computes a Karcher mean and creates a boundary based on a scalar multiple of the standard deviation, taken away from the Karcher mean. Growth curves are aligned using the average reparametrization function, $\bar{\gamma}$ (such as in Distance Method 2), and the aligned growth curve is compared to the created boundary.

Method 3: This method incorporates Distance Method 1 to find distances between growth curves and the Karcher Mean. It then uses a threshold value to determine the classification of the chick. In particular, the threshold value is based on the average of all of the distances added to a scalar multiple of standard deviation of the distances.

Method 4: Similar to the previous method, however this time, Distance Method 2 is used to compute the distances before classification.

We want to test our classification algorithms by comparing the predicted health values to a ground truth. In collaboration with the Kākāpō Recovery team, we devised a list of general health statuses to use as a ground truth. The samples are then split into groups based on the sex, ripe-Rimu and hand-reared factors. For each group, we start by selecting a sample of growth curves that will be used to compute the Karcher means needed for Methods 2-4, as a *training* set. This training sample will contain growth curves of similar lengths (in ages), ranging from 0-75 days of age. Thereafter, we create a separate sample of growth curves, a *testing* set. In this second sample, growth curves of all lengths are included, but still with a minimum age of 0 days and a maximum age 75 days[132]. This means that shorter curves (for example, only a few days long), may also be included. From here, the next step is to create the boundaries and distance thresholds. We test various values for the parameters involved for Methods 2-4[133]. To do this, for each parameter, we compute a score based on the classification results, and the ground truth. Henceforth, we output the results that achieved the highest score.

[132]Although in this case we are simply considering growth curves of ages up to 75 days, in general we may want to alter the minimum and maximum age, for example, if we wanted to exclude the weights in a chick's first week, and so on.

[133]Note that for Method 1, in this example, training is bypassed, as is parameter testing. Instead, the mean and 20% boundary are created using the testing sample. One reason for this was because the size of the training samples used in the other methods is rather small. Secondly, by computing the mean from the test set, our method is more in-line with the techniques currently used by our collaborators.

To calculate classification scores, we must take into consideration the implications of false positives and false negatives:

False Positives – the incorrect classification of healthy chicks as unhealthy. When a chick is classified as *unhealthy*, it will be temporarily hand-reared or moved to another nest. However, if the classification is a false positive, this may cause a waste in resources that could have otherwise been used for an unhealthy chick who required help.

False Negatives – incorrectly classifying unhealthy chicks as healthy. This can have dangerous implications that can leave a genuinely unhealthy chick untreated.

When it comes to predicting the health of a species in conservation research, the implications of false negatives are far more severe than the implications of false positives. For this reason, here we will not employ a score that weighs the rates of these equally. As we are interested in a well-regarded scoring technique that is well-used within the machine learning community yet does not weigh rates equally, this naturally leads us to the generalisation of the standard F_1-score incorporated in previous projects, i.e. the F_β-score (where β trades off the relative importance of precision against recall).

$$F_\beta(P, R) = (1 + \beta^2) \times \frac{PR}{P(\beta^2) + R} \tag{5.11}$$

where P and R are the precision and recall scores, as defined in Equations (4.3) and (4.4) respectively. Using this type of score, as opposed to an F_1-score, grants us the power to decide on the level importance each component has. In our case, it is beneficial to weight the recall lower than the precision, since it is important that we don't miss out on classifying any unhealthy chicks. For this reason, we set β to be a low number.

In order to decide on our weight parameter, β, we performed classification tests on all groups and subgroups of datasets (i.e., based on the *classes* of the chicks), using not just the four methods detailed here, but other variants of Method 1, that were based on different percentage values. We then then computed the scores with different values of β and analysed the results, in consultation with the Kākāpō Recovery team. Thus the F score parameter was then fixed for the remainder of the analysis, at $\beta = 0.15$.

Figure 5.5 shows confusion matrices from our experiment. Here we see the top results of

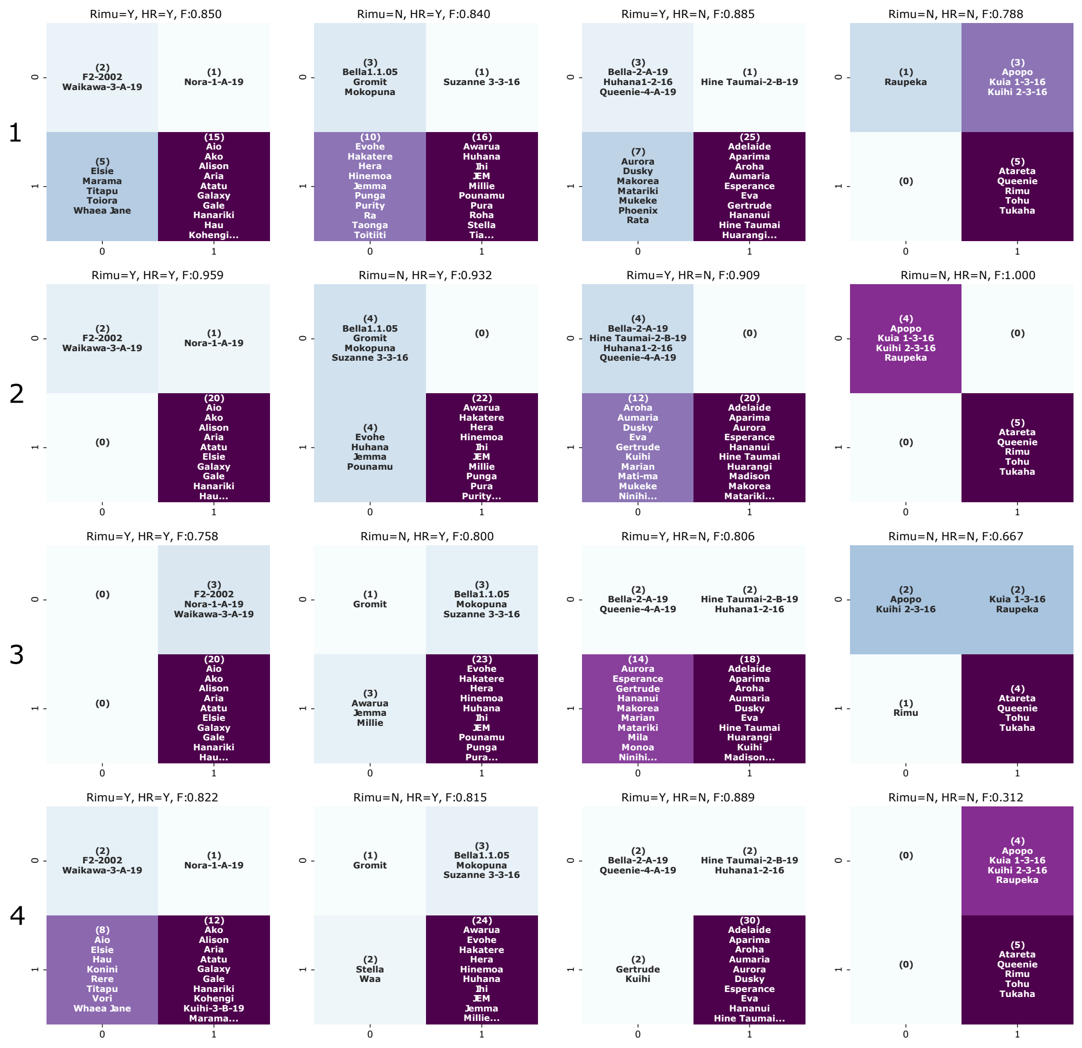


Figure 5.5: Confusion matrices based on the top classification results per method. The results shown here are from a sample of female chicks, aged 0-75 days, where the samples are based on the ripe-rimu and hand-reared factors. The individual plots are heat-maps, that display the F-scores in the titles, and the names of some of the chicks that were classified, with the total count of chicks in each category displayed above. Here, 0 and 1 stand for *unhealthy* and *healthy* respectively and the true classifications are on the y-axis, whilst the predicted classifications are on the x-axis. Moving clock-wise, starting from the top-left square on a plot, the results show the proportion of *true negatives*, *false positives*, *true positives*, and *false negatives*, with darker colours representing greater proportions. The results here show us that Method 2 outperformed all other methods, with an average F_1-score of 0.95 across the four samples. Interestingly, the two distance-related methods (3 and 4) performed the worst, with their F_1-scores averaging 0.76 and 0.71 respectively. Meanwhile, the original Method (1) had a more decent average F_1-score of 0.84, though this can be down to the method's over-tendency to label samples as *unhealthy*.

each method, for each group of female chicks. In this example, we tested out 13 evenly-spaced values for the standard deviation scalar, k, used in the boundary for Method 2, where $1.5 \leq k \leq 4.5$. Meanwhile, for the standard deviation scalar used in the distance threshold, m, for Methods 3 and 4, we tested 11 evenly spaced values, where $0 \leq m \leq 5$. We remark that the options for the paramaters k and m are different as they are based on different values, with k relating to a standard deviation of the Karcher mean, and m is associated with the standard deviation of the *distances*.

The results here show us that Method 2 always came out on top. Interestingly, despite the simplicity of Method 1, it performed remarkably well, and outperformed some of the other methods. Nonetheless, it was Method 2 that was the only method to classify almost all of the unhealthy chicks correctly in each group. These results were echoed in the tests made on the male kākāpō chicks sample.

5.1.8 Classifying Growth Curves with Karcher Means

The plots in Figure 5.5 identified our best method. Here, we will take a deeper look into the algorithm that defines this method, which comprises of Karcher Means and a boundary to classify the health of kākāpō chicks.

Sampling Growth Curves

Kākāpō chicks are weighed regularly, but that does not mean that they are all weighed an equal amount of times. Our growth curves are each comprised of a varying amount of points. Meanwhile, the Karcher mean computations require curves to be of an equal number of points. Henceforth, this forces us to *regulate* the points, as otherwise, we would have to change the Karcher mean. We therefore resample the growth curves for the computational steps in our algorithm.

Very often, in order to sample curves to have a certain number of points, an interpolation is performed on the curve, such as a linear interpolation on a fixed set of x-values[134]. However, to stay as true as possible to the original growth curve, we opted for a slightly different sampling method. Fundamentally, the sampling function involves creating an initial template based on the first and last value of the growth curve. Hereafter, the

[134]This is especially inconvenient for us as the chicks are weighed at unequal ages.

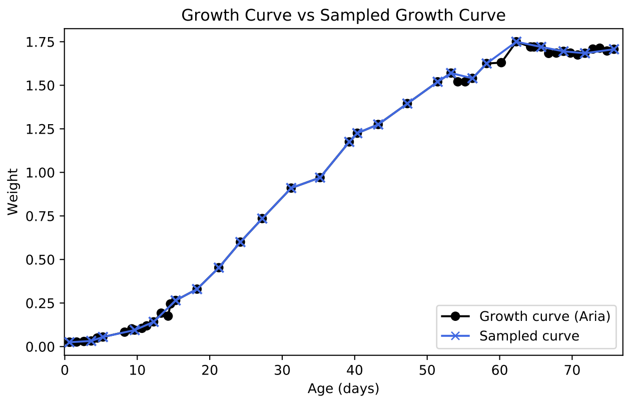


Figure 5.6: Example of a growth curve of a female chick (Aria) aged between 0-75 days, with 50 points (circled, black). The plot also features a sampled growth curve containing 25 points (crossed, blue). The plot shows how similar the sampled growth curve is to the original, despite the significant *cut* in the number of points.

closest values in the ages of the growth curve to the template are chosen, and hence the respective weights to those ages. This function is briefly outlined in Algorithm 11, and an example plot can be seen in Figure 5.6.

Outside the Boundary

On numerous occasions we have discussed growth curves falling outside of a boundary, but have scarcely described what that truly means.

Figure 5.7 shows the aligned growth curves of two chicks (*Aria* - ground truth classification: *healthy*; *F2-2002* - ground truth classification: *unhealthy*) along with a boundary section (magenta). Here, one curve is fully inside the boundary, but the other is not, although its majority is in fact inside. After toying with the idea of using a majority ratio to determine the classification of a growth curve based on a boundary, we opted to use a consecutive point approach instead. Thus if a chick had K consecutive points outside of the boundary, it will be classified as *unhealthy*. In this particular example, this would be the case for the latter chick if $K \leq 13$.

Similar to previous experiments, we tested out various values for K in order to get an understanding of the effect it has on the classification algorithm. We used the top method (Method 2), with varying values for the standard deviation scalar, and output the results

Algorithm 11: Growth Curve Sampling

`sample_growth_curves(T,F,N=40)`

Aim:

Sample a growth curve to have a `N` points.

Input:

A growth curve of a chick with `T` time-points and weights `F` at time `T`.

Code:

1. Create a template: `template = numpy.linspace(T[0],T[-1],`N`)`.

2. Set `newIndex = [0,length(T)`-1`]`.

3. Set `missingValues = []` and find closest time-points to template:

```
for $\imath \in N$ do
    index = numpy.where((T $\leq$ template[$i$]) & (T$\geq$ template[$i-1$]))
    if length(index)$>0$ then
        index = [$p \mid \forall p \in$Index, where, $p \notin$ newIndex]
        if length(index)!=0 then
            newIndex.append(index[-1])
        else
            missingValues.append(i)
        end
    else
        missingValues.append($i$)
    end
end
```

4. search through `missingValues`, using larger neighbourhoods to try and find more index values to add to `newIndex`. If values still cannot be found, then we arbitrarily select k values from `T` and append them to `newIndex`, where k is the final number of missing values.

5. Let `Ts = T[newIndex]` and `Fs = F[newIndex]`

`return Ts` - sampled `T`

`return Fs` - sampled `F`

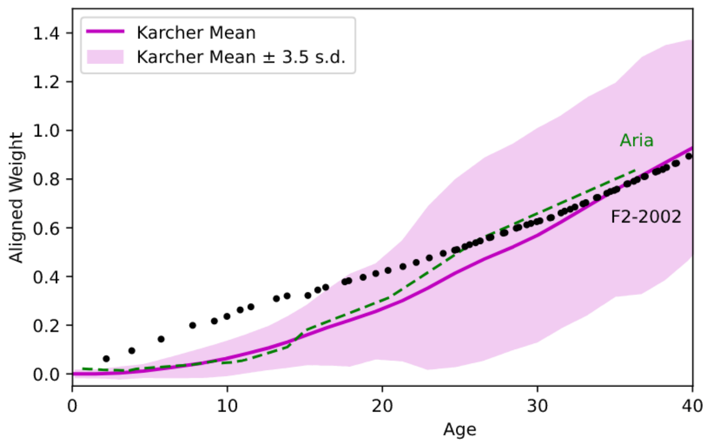


Figure 5.7: An aligned Karcher mean (magenta) and a boundary based on the Karcher mean ± a multiple of the standard deviation. The aligned growth curves of two chicks are shown, Aria (dashed, green) and F2-2002 (dotted, black).

with the highest $F_{\beta=0.15}$ score. Figure 5.8 displays the confusion matrices from ripe-Rimu chicks aged, 0-75 days. In this example, there are six values for K, where $3 \leq K \leq 15$. Intriguingly, the results show differing outcomes. Whilst they unilaterally perform the poorest with $K = 3$, their best results all differ. For instance, the classifications for the female, non-hand-reared chicks improve with $K = 12, 15$ but the results from the same K on the male, non-hand-reared chicks were remarkably worse, the optimal here coming from a lower $K = 9$.

At the end of the day, we want to create an algorithm with the highest chance of correctly classifying the health of chicks. Although it is easy to say that in order to obtain the most *accurate* classifications, we can utilize the results of Figure 5.8 to optimize the K parameter for each sample, in actual fact, the decision is not so simple, as we must consider the impact on conservation. For example, the greater the value of K is (the number of consecutive days a chick falls outside a threshold), the longer we will have to wait in real time, before making a prediction on the health of chicks. Thus, we do not want the parameter K to be too high. This is why, as with many practical applications, it is important to work with the experts, in order to make such decisions.

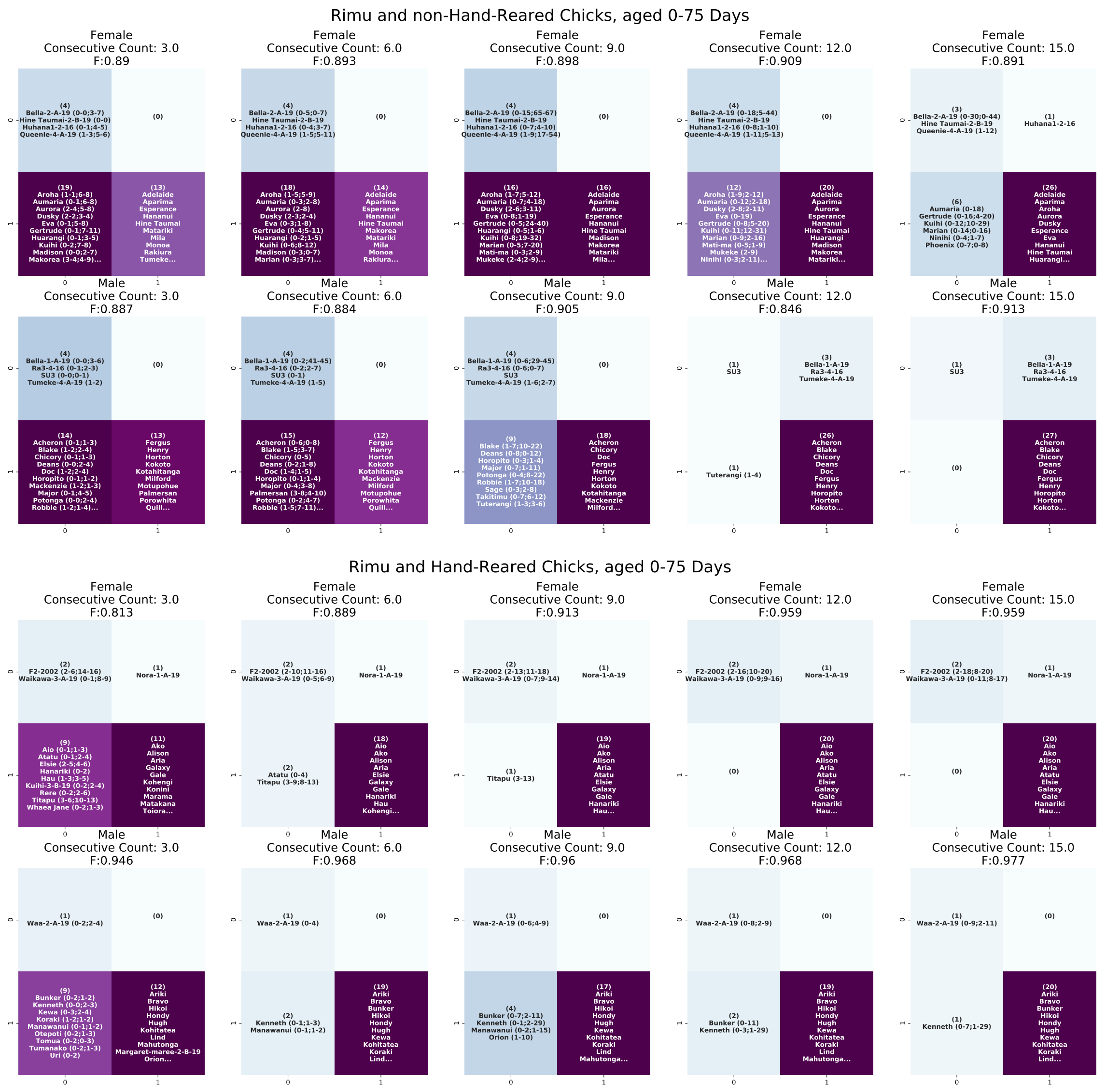


Figure 5.8: Confusion matrices for the comparison of different values of K (labelled here as *consecutive count*). Each confusion matrix is a heat-map representing (clock-wise, from the top-left corner) the chicks that were classified as *true negatives*, *false positives*, *true positives*, *false negatives*, where the darker the colour is, the greater the proportion is. Inside the boxes, there are the names of some of the chicks within these proportions. Here we see the results from chicks born during ripe-Rimu season. As before, *unhealthy* and *healthy* classifications are labelled as 0 and 1 respectively, and F is the $F_{\beta=0.15}$ score. Additionally, the numbers written next to the names of the chicks that were classified as *unhealthy* represent the first and last segment of ages where the growth curve fell outside of the boundary. For example, in the last plot (bottom, far-right), we see that the chick *Waa-2-A-19* had consecutive points between ages 0-9 and 2-11 days outside of the boundary.

Precautionary Steps

Classification can make a big impact on conservation. Therefore, when it comes to kākāpō conservation, it is more important than ever to get it right. Hand-rearing an unhealthy chick (or moving it to another nest) can undoubtedly save its life, ergo a misclassification of an unhealthy chick could have grave consequences. Concurrently, over-classifying chicks as unhealthy can also lead to harmful impacts, such as a lack of resources for chicks who genuinely require more help, or even negative imprinting[135]. For more details on the effects of hand-rearing, we refer the reader to (Eason and Moorhouse, 2006). Henceforth, it is vital to provide an additional safety net to the classification algorithm, just in case, even if they are rarely required.

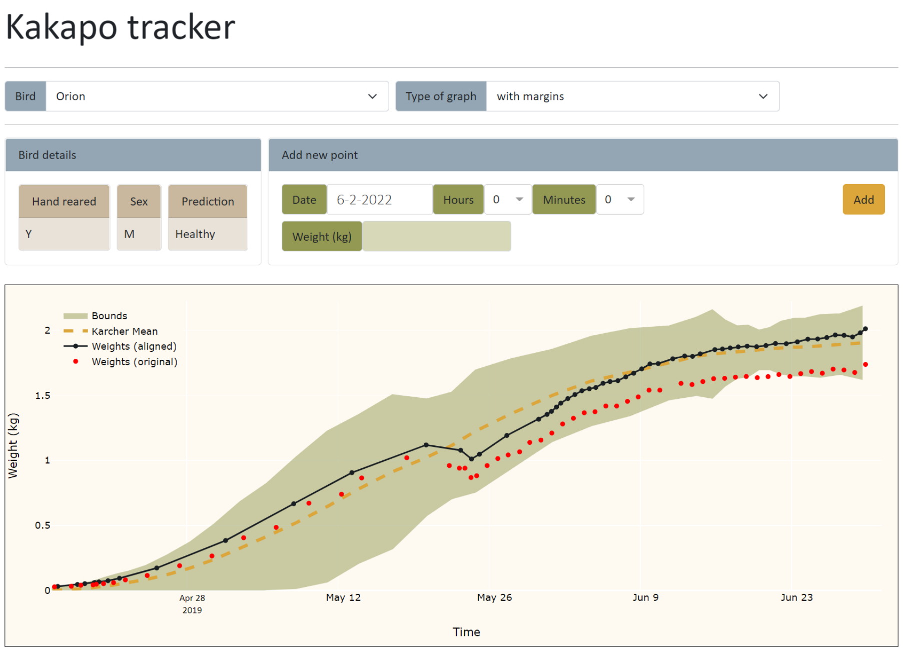
1. **Misclassifying Unhealthy Chicks (*potential false negatives*):** If after the initial classification (based on chosen method) a chick has been classified as *healthy*, we perform an additional check, in order to reduce the risk of a sample being a false negative. To do this, we compare the growth curve to a new boundary (lower bound), this time based on the standard cross-sectional mean. This is similar to the original method of detecting underweight chicks, however with a much lower boundary, where the default is currently set at mean−75%. Thus, if a growth curve is lower than this new extreme boundary, then it is reclassified[137] as *unhealthy*. Note that this safety net is only applicable in extreme cases and it is rarely utilized. Nonetheless, it remains in the algorithm, just in case, the original methods fail to correctly identify the health of a *severely* underweight chick.

2. **Misclassifying Healthy Chicks (*potential false positives*):** If a growth curve has been classified as *unhealthy* in the initial classification, in order to decrease the risk of it being a false positive, we perform an additional test. Here, we compute the ratio of points in the latter half of the chick's growth curve that are above the standard cross-sectional mean. If this ratio is greater than some threshold (currently set at 0.5), then the chick is reclassified as *healthy*. Importantly, it must be noted that there are a few caveats applied here. Firstly, if a chick is on a negative trend, determined by its gradient at the last three days of its growth curve; or if

[135] Sirocco[136] the kākāpō is a notable example of a chick who imprinted on humans after intensive hand-rearing early on in his life.

[136] https://www.doc.govt.nz/nature/native-animals/birds/birds-a-z/kakapo/sirocco/

[137] At present, this additional filter is only applicable on chicks who are ≥ 10 days old, as original boundaries in the first week of a chick's life are finer.


Kakapo tracker

Bird: Orion
Type of graph: with margins

Bird details

Hand reared	Sex	Prediction
Y	M	Healthy

Add new point

Date: 6-2-2022
Hours: 0
Minutes: 0
Weight (kg):
Add

Figure 5.9: Example shot of the kākāpō health classification program, developed by Giotto Frean. A chick can be selected and its health predicted. Furthermore, new points can be added to the growth curves of the chicks. Note that this page is connected to the live data that is tracked and collated by the Kākāpō Recovery group.

a chick's greatest age is much lower than the maximum age[138] we have set in the mean template, then the chick's health is not reclassified. This safety net is not too common, but out of a test on 50 chicks, the safety net was incorporated on the classifications of three of them.

5.1.9 Summary & Significance in Conservation

Throughout this project, the aim has been to create an algorithm to aide with the conservation of these marvellous birds. By creating a web-based platform, we unlock the true functionality of our algorithms and in turn, enable the team at Kākāpō Recovery to take full advantage by utilizing the classification tool in real-time.

[138]We require the maximum age in the growth curve to be greater than a fifth of the maximum age in the mean template for this particular extra filter to be applied.

Kākāpō Chick Health Detection Tool

The results from our research revealed that the top method for chick health classification was with Method 2 (see Figure 5.8 for example) combined with the safety-net features discussed earlier. By writing a series of algorithms based on this method and the necessary Python code (examples of which can be seen in Appendix A.1), with the help of a web developer, Giotto Frean, we created an online tool that can be used to monitor the health of kākāpō chicks.

Figure 5.9 shows an example of the web interface of the classification program. To make it as simple as possible, the only mandatory field is the chick name. By selecting a chick, a growth curve of the chick is shown along with the chick's predicted classification. Furthermore, the program is linked to the live data that the Kākāpō Recovery group track, and can provide predictions on-the-go. Moreover, as it is connected to the live database, the program can identify the sample of the chick (i.e. based on its sex, and ripe-rimu / hand-reared status), display it and use it optimize parameters.

In collaboration with Kākāpō Recovery, we designed a scheme aimed at enabling the classification tool to run as smoothly as possible. To do this, we pre-computed values whenever necessary and categorised the data into three sections:

1. **Meta details** – the weights of chicks at given timestamps, as well as the additional parameters we used to sample chicks (sex, ripe-rimu / hand-reared status).
2. **Karcher means** – to avoid computing the Karcher mean every time a prediction is made, we pre-compute the means on a daily basis, for each sample group. This dataset also involves the standard deviation and the average warping function, $\bar{\gamma}$, as utilized in Method 2 (see Section 5.1.7).
3. **Optimal parameters** – The optimal parameters for each sample group, such as the standard deviation scalar, or the number of consecutive points we use when defining the health. Unlike the Karcher means, to reduce the time spent on optimization, these values are updated less regularly.

Summary

Until now, methods of elastic shape analysis had seldom (if ever) been applied to conservation projects. Here, we have seen that such methods can not only be applied, but can be incorporated successfully in creating useful tools, that have the potential to work better than current methods that conservationists often employ.

Thus far, we have focused on young, pre-fledged chicks, therefore, our next step is to create a technique that can be used to classify older, fledged chicks. To do this, we will train our algorithms on older chicks. This will involve working directly with the Kākāpō Recovery group, once again, in order to obtain *ground truth* health classifications. Furthermore one possible challenge that may arise when working with post-fledged chicks, is based on the *historical* data the chicks possess. Thus, when performing classification here as well as additional statistical analysis on the growth curves, we must study the question *how far back should we go in the chick's growth curve, to include it in the classification?*

Throughout this section, we have obtained decent classification results, and thus created a novel platform to be used by the Kākāpō Recovery group, in order to monitor chicks. Soon, after the next breeding season, our methods will be utilized to monitor the next group of chicks. And when this happens, we can begin to see the true performance of our methods, as well as the potential of the applications of shape analysis in conservation. And if our techniques prove even just a little helpful in the monitoring of these incredible, endangered species, then we have made a true difference, and can be proud.

Figure 5.10: A young kākāpō roaming around. Photo taken by A.Digby.

5.2 Kiwi Call Identification

For millions of years, New Zealand flourished with an abundance of birds. With no mammalian predators, birds thrived, and evolved into some of the most unique and wonderful species on the planet. Many are/were endemic to New Zealand; from the flightless South Island Giant Moa (the tallest known bird species to have existed), to the world's only alpine parrot, the Kea. However, with the start of human settlements in New Zealand in the last millennium, everything changed. Over-hunting, habitat-destruction, and the introduction of invasive species led to the unfortunate extinction of many of these wondrous birds, such as the mighty Haast's Eagle, the elegant Huia, and the distinctive Whēkau (Laughing Owl). This has motivated the need for conservation projects and bodies[139] aimed at protecting the remaining, remarkable birds of New Zealand from similar, dire fates. Examples of conservation efforts range from simply erecting pest-proof fences around certain habitats, as described in (Burns et al., 2012) to nationwide projects such as Predator-Free Wellington[141], the Kiwi Recovery Plan[142], and Predator Free New Zealand 2050[143].

5.2.1 Introduction to Automatic Birdsong Recognition

It is no secret that birds are rather talkative, *to say the least*, and are often identifiable from their song. Thus, ornithologists and others alike, have used this as the basis for monitoring species for some time. A common approach is the *5-minute call count*, which, broadly, involves spending 5-minute intervals identifying as many birds as possible in certain locations. Further details and similar methodologies are described in (Ralph et al., 1995). Though the premise is simple, these types of listening methods have their drawbacks, such as being time-consuming, susceptible to human errors, and biased towards species that are not afraid of humans and/or those that live in accessible territories. Nonetheless, they are commonly used to monitor species in New Zealand today, including in volunteer projects (Peters et al., 2016).

[139]For more details regarding some examples of bird conservation projects, check out New Zealand's largest independent, conservation organization, Forest & Bird[140].

[140]`https://www.forestandbird.org.nz/`

[141]`https://www.gw.govt.nz/environment/pest-management/predator-free-wellington/`

[142]`https://www.doc.govt.nz/globalassets/documents/science-and-technical/tsrp64entire.pdf`

[143]`https://predatorfreenz.org/the-big-picture/our-vision-taonga/what-predator-free-2050/`

In recent years, we have seen the rise of cheap, automatic, acoustic recorders that have revolutionised birdsong monitoring. These devices offer the prospect of monitoring methods that are far less obtrusive than in-person listening techniques, henceforth there are now thousands of acoustic recorders deployed in New Zealand. The potential for this data is enormous, particularly with regards to automatic birdsong recognition. This can prove a highly useful tool in conservation, for example in estimating species abundance[144].

Visualising Birdsong

When analysing audio data from recorders, actual bird-calls are rather infrequent overall. Therefore, instead of analysing audio in real-time, it is often easier to employ a visual representation of the recording. A common method in visualising sound data is with histograms called *spectrograms*[145]. An example of a spectrogram plot from a recording of a male Kiwi can be seen in Figure 5.11. These spectrograms can be thought of as heat-maps, depicting the frequency (*vertical axis*) and power (*intensity*) of a recording, over time (*horizontal axis*).

By employing spectrogram representations of audio data, we expand the ways in which we can automatically analyse birdsong. For example, in (Kahl et al., 2021), image-processing tools were combined with a deep artificial neural network, to classify bird species from spectrogram data. In addition to the usual dependence of large training sets in deep learning, these methods are also susceptible to noise. This has led to literature focusing on denoising birdsong sound data, such as (Priyadarshani et al., 2016), that can be used as a pre-processing step for further analysis. Alternatively, by working with spectrograms, we can turn to the field of shape analysis.

Spectrogram Shapes

Shapes can certainly be found in spectrogram plots. As seen in Figure 5.11, a spectrogram displays the components of a call: a fundamental frequency together with a set of har-

[144]Estimating populations via acoustics is particularly important for most bird species, as birds are genereally more easily heard than seen. Acoustic methods are even more vital for certain New Zealand bird species, where the birds live in bushes, are nocturnal, and are generally well-camouflaged. Though we note that there are some exceptions where image-based recognition is helpful such as the research in (Fretwell et al., 2017), regarding albatrosses.

[145]We note that we will elaborate on the computation of such graphs later on in this section.

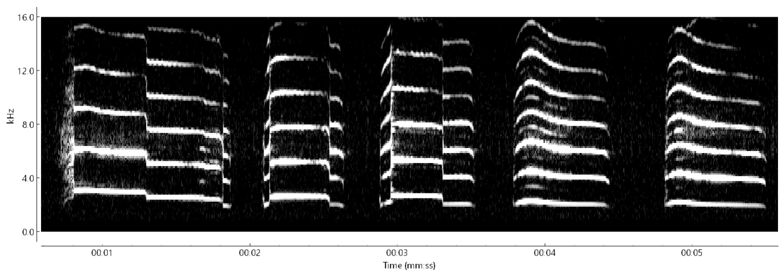


Figure 5.11: Spectrogram from a recording of a male kiwi call. The y-axis corresponds to frequency, whilst the x-axis corresponds to time. The intensity of the heat-map corresponds to the power in that frequency.

monics[146]. These harmonics, and their number, provide a host of information, from the species anatomy and power, to the distance from the recorder. By gathering information from the call, we can analyse birdsong from recorders, in order to identify species. Commonly, this is done using machine learning algorithms, such as support vector machines to recognise individual birds in (Cheng et al., 2012), neural networks for classification in (Qian et al., 2015), or a Markov model approach in (Lakshminarayanan et al., 2009). However, what we see in spectrograms are simply shapes. This is a *shape* problem; hence the question arises, *why not use shape methods?*.

At present, shapes are seldom studied when it comes to the analysis of audio data. However, there are few exceptions, for example, the literature found in (Rocha and Romano, 2021) and (Norman et al., 2013), where Geometric Morphometrics techniques, such as a semi-landmark approach, were employed to study 3D spectrogram data. But the focus in the aforementioned literature are not on individual curves. Thus the question arises: *can we use shape analysis on the true curves?* As elastic shape analysis has not been applied to audio data[147], this motivates us to examine the possibility of applying such methods for automatic birdsong classification.

5.2.2 Project Introduction

We are interested in the applications of elastic shape analysis to spectrogram curves from audio data. Therefore, we have began a project in collaboration with AviaNZ[148], to

[146]Note that we will go into more details about such frequency curves later on.
[147]Excluding very difficult applications to SONAR data, as seen in (Tucker et al., 2013).
[148]`https://www.avianz.net/`

Figure 5.12: Juvenile Little Spotted Kiwi at Zealandia Ecosanctuary, NZ.

investigate whether shape methods can be used to identify individuals within a species.

AviaNZ

AviaNZ is a collaboration between a variety of experts ranging from bird ecologists and conservationists to mathematicians and data scientists. To tackle the questions regarding species classification, the team designed a software package, called AviaNZ, to analyse birdsong; incorporating both manual annotations, as well as automated species detection. For more details regarding the software, we refer the reader to (Marsland et al., 2019). The team are involved in numerous conservation projects which involve field-work and research on automatic species classification using the AviaNZ software.

Kiwi Birdsong Data

Our global aim is to develop methods that can be applied to the acoustic classification of kiwis. But to begin with, we focus on one species: the North Island brown kiwi (*Apteryx mantelli*). As our AviaNZ collaborators already possess a large amount of calling data from this species, this seemed like a logical choice.

Kiwis are flightless, nocturnal birds that are endemic to New Zealand. They are easily recognisable by their long and thin beaks (which, with nostrils attached to end, lead to their great sense of smell), and notable for their egg size (which is the largest in proportion to body size for any bird species); and thus, kiwis make for a rather adorable national icon. Furthermore, though once abundant throughout most of New Zealand, four of the five kiwi species are now classed as *vulnerable* according to the International Union for Conservation of Nature[149]. For this reason, there have been numerous conservation projects to help save the kiwi. In particular, the Department of Conservation[150] has set up various sanctuaries, allowing kiwis to roam free without the threat of predators; for example, Zealandia[151] in Wellington, where the photo in Figure 5.12 was taken.

Kiwis have rather distinctive calls, which are made up of syllables containing multiple harmonics[152]. The males have a periodic high-pitched call, whilst the female kiwis have a lower and more rasping croak-like call. We focus on the male kiwi calls, where a spectrogram example can be seen in Figure 5.11.

Project Aims

The goal is to design a framework that extracts curves from spectrograms, computes *distances* between *sets* of harmonics, and classifies individuals within a species, by, primarily, comparing the shapes of the harmonics that the individuals produce in their calls. We begin by focusing on a dataset of recordings taken from 4 male North Island Brown Kiwi, who were fitted with individual recorders. Furthermore, to develop our methods on this dataset, our project is split into three sections, which bring together tools from signal and image processing, elastic shape analysis, and machine learning:

1 – Call Extraction from Spectrograms The first step is to extract calls from spectrograms, i.e. the curves seen in Figure 5.11. To do this, we use a combination of signal processing methods (which we will go into more detail about in the next subsection) and contour extraction methods, as discussed in previous chapters.

2 – Differences in Harmonics Once we have curves describing calls, we can begin to make comparisons between them. This involves using shape analysis to compute pairwise distances between curves, as well as computing differences based on additional information such as the average frequency or length of the harmonic.

3 – Machine Learning Classification Finally, we use the information found in the previous step and train a machine learning classifier to classify the kiwi calls to the

[149] https://www.iucn.org/it
[150] https://www.doc.govt.nz/
[151] https://www.visitzealandia.com/
[152] Note that we will come back to the structure of kiwi calls later on in this section.

individual.

5.2.3 Signal Processing Fundamentals

In everyday life we don't often think about *visualising* audio. And in those moments when we do ponder about *plotting* audio, it may be more common to picture waveform plots, instead of spectrograms, as they're often seen in logos and popular culture (take for instance, the album cover for the 2013 album, *AM*[153], by the indie rock band Arctic Monkeys). Meanwhile, harmonics, or in general, frequency waves, are also sometimes referenced in popular culture, albeit more subtly, such as the logo for the streaming provider, Spotify[154]. *But what do these waveforms and frequency wave curves mean?* In this section, we go through the fundamentals of signal processing tools that allow us to visualise and mathematically describe audio data.

Visualising Audio

An audio file is a digital representation of sound made by sampling the air pressure at regular intervals. This is an audio time series which can then be plotted as a *waveform*, as in the top plot of Figure 5.13. Waveforms represent the amplitude of a sound wave over time. Thus, if the waveform is at 0, this can be interpreted as silence.

In order to get a better understanding of an audio signal, we want to visualise not just the amplitude over time, but the frequency too - this brings us onto spectrograms. For this we need frequency information.

To compute a spectrogram from an audio signal, the audio is first split into windows, and a discrete Fourier transform is computed for each window:

$$F_k = \sum_{n=0}^{N-1} a_n \exp \frac{-2\pi i}{N} kn \tag{5.12}$$

where $\{F_k\}$ is the spectrum of the frequencies, and $\{a_i\}$ are the discretized amplitudes sampled from the audio file. In order to compute the discrete Fourier transforms, DFTs, a Fast Fourier Transform, FFT, algorithm is used, such as the algorithm described in

[153] https://en.wikipedia.org/wiki/AM_(Arctic_Monkeys_album)#/media/File:%22AM%22_(Arctic_Monkeys).jpg

[154] https://www.spotify.com/us/

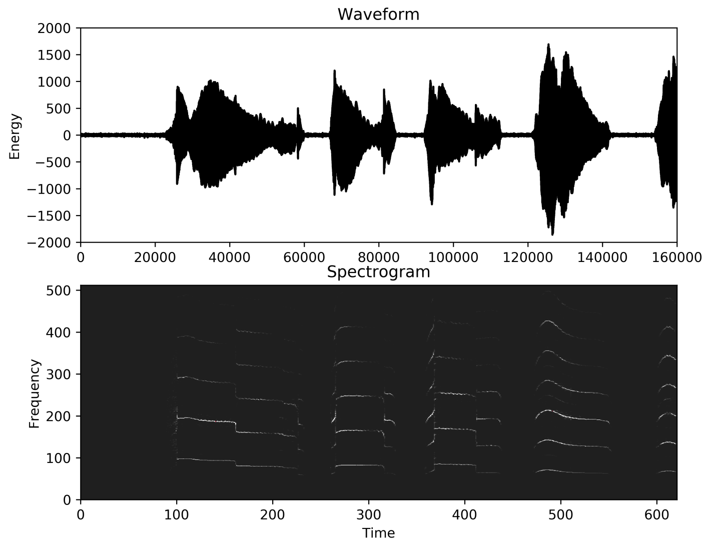


Figure 5.13: A 6-second clip of a recording from a male kiwi call. Top: a waveform plot over time. Bottom: a spectrogram from the audio.

(Cooley and Tukey, 1965). The output of the FFT is the amplitudes of different frequencies.

We note that Fourier transform analysis assume that signals continue infinitely in time. Thus, unless the signal in each window is aligned with the entire sampling time, the DFT will simply produce a spike; which is not so helpful. To overcome this, a *window function* is used that decays to 0 outside some range. A commonly used function is the *Hann function*:

$$\omega(n) = \frac{1}{2} - \frac{1}{2}\cos\frac{2\pi n}{M-1} \tag{5.13}$$

where $0 \leq n \leq M-1$ and M is the chosen number of points in the window[155]. Throughout this project, we use this Hann function for our spectrograms. For more details regarding the Hann window function and other window functions, we refer the reader to the book (Prabhu, 2014).

The final output is a two dimensional matrix containing the magnitudes of frequencies over time. This can then be plotted as a heatmap, i.e. a spectrogram. An example of a spectrogram can be seen in the lower plot of Figure 5.13, where the whiter the colour is, the stronger the frequency.

[155]We note that in this project, we stick to a 256-sample Hann window.

Figure 5.13 demonstrates how we can plot sound such as kiwi calls collected from recorders. We see that the spectrogram plot is far more helpful than the waveform plot, as we can get an idea of the frequencies involved. The next step is to define the dominant frequency that those harmonics are based on.

Dominant & Fundamental Frequencies

A spectrogram displays the spectrum of frequencies from an audio signal. Some of these frequencies may be stronger than others, such as those represented by the brightest curves displayed in Figure 5.13. The question of how to define these *bright* curves brings us to the topics of dominant frequencies and fundamental frequencies.

As described in (Telgarsky, 2013), a dominant frequency is the frequency carrying the maximum energy with respect to other frequencies found in the spectrum. Meanwhile, the fundamental frequency is the lowest frequency that has a peak among the other frequencies. Colloquially, this can be thought of as the brightest lowest frequency wave seen in a spectrogram; as well as the frequency wave which the harmonics in a call are based on. When the fundamental frequency has the largest amplitude, then it is considered to be both the fundamental and dominant frequency.

The dominant and fundamental frequencies play a huge part in audio analysis and signal processing. This is no surprise as they are the prominent frequencies our ears perceieve when listening to audio signals, (Bruce and Marilyn, 1993). By analysing these frequencies, we learn a lot about the audio signal in question; for this reason they are often used in applications to signal processing, such as in (Pal et al., 2018), where the fundamental frequency is used to detect synthetic speech.

Estimating the fundamental frequency is a significant topic in signal processing, as emphasised in (Gerhard et al., 2003). Though it is not always a simple task, especially when the signal is noisy, there exist algorithms that take on the job rather successfully, such as the well-known YIN algorithm (De Cheveigné and Kawahara, 2002). More often than not, these algorithms, including YIN, involve the computation of the autocorrelation;

which, in signal processing, is defined as:

$$R_{xx}(\tau) = \sum_{n=0}^{N-1-\tau} x(n)\overline{x(n+\tau)} \tag{5.14}$$

for a finite, discrete-time signal $x(n)$ of size N. Broadly, by finding the maximum of the autocorrelation function of a signal, $x(n)$, we can estimate the fundamental frequency.

We will see how the frequencies aide us in extracting *curves* from spectrograms, in order to define kiwi calls. Finally, for more details on the evaluation of techniques used to estimate such frequencies, we recommend (Gerhard et al., 2003).

5.2.4 Extracting Curves from Kiwi Calls

In the simplest terms, we want to compare the calls of individual kiwi birds. But to do that, we must first define what we mean by a *call*.

Defining Kiwi Calls

The call of male kiwis can be described as a shrill, repeating trill. Consequently, in high-quality recordings, this results in spectrograms that contain clear harmonics, of generally three or more variations, repeated over time. Other species, especially those whose calls also consist of distinctive high-pitched trills, such as the Ruru (an owl, also known as a Morepork), produce similar spectrograms. For a detailed analysis on the vocalisation of a kiwi calls (for Little Spotted Kiwi, specifically), we refer the reader to (Digby et al., 2013). Additionally, for an overview of calls from a variety of New Zealand bird species, we recommend the following study on the environmental effects of calls of twenty different New Zealand birds, obtained through automatic recorders, in (Priyadarshani et al., 2018). To hear snippets of calls from kiwi as well as many other species in New Zealand and elsewhere, check out the AviaNZ *cheat sheet*[156].

Spectrograms display a host of frequency curves for all animals. In general, most animals deliberately make the lowest frequency, called the *fundamental frequency*, whilst other frequencies appear as a result of vibrations on the vocal chords. For birds however, due to the nature of their syrinx (vocal organ), they can emphasise multiple frequencies, where the one with the most *power* is labelled the *dominant frequency*. For this reason,

[156] https://www.avianz.net/index.php/resources/cheat-sheet/diurnal-birds

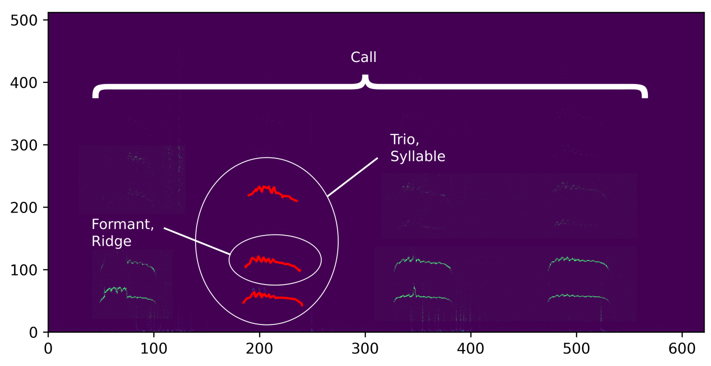


Figure 5.14: A spectrogram from a recording of a male kiwi call. The faint green curves represent the frequencies with the most *energy*. In order to label the terms we use in this project, we have highlighted three curves in red. Distinct individual curves representing the harmonics can be called *formants* or *ridges*. A *syllable* is the grouping of those formants, i.e. *harmonics*, into a single piece of a call, separated by silence. In this specific project, we look at syllables that contain three formants, thus the informal term, *trio*, may also be used for the specific syllables in this project.

we can see harmonic curves at different frequencies on spectrograms, as in Figure 5.14.

In Figure 5.14, we show a spectrogram computed from a 5-second recording of a male kiwi call. As before, the lighter the colour, the stronger the frequency. Here, we have highlighted three distinct curves (originally in a faint blue) in red for emphasis. The curves are harmonics, which, here, we will refer to as *formants* or *ridges*. These formants are variations of the same curve. Henceforth, we describe the group of all such variations as a *syllable*, and describe a *call* as a collection of *syllables*, with silence in-between, as labelled in Figure 5.14.

We note that in general, male kiwi calls contain multiple harmonics per syllable. The number of *strong* formants can certainly vary but in our study, observations showed that this number was generally greater or equal to 3. Therefore, in this project, for the purpose of simplicity and consistency, we restrict our syllables to exactly three formants. These groups may be referred to as a *trio*.

Extracting Calls from Spectrograms

We wish to extract the curves (represented by x and y coordinates), that form a call of a kiwi from a spectrogram. For the most part, we treat the spectrograms as images and utilize contour extraction tools seen in previous sections as well as the Image Processing chapter. Though, as you will soon see, in one part of the curve *extraction*, we do not treat the spectrogram as just an image, and instead, we compute the dominant frequency curves, based on the recording and its spectrum of frequencies. Overall, we employ a five-stage process in order to form a dataset of kiwi calls from recordings. We briefly describe the aforementioned process in the next paragraphs and subsequently outline the Python code we designed to complete the tasks.

Step 1 – Dominant Frequency Curves The motive of this first step is to gain an idea about where the harmonics lie in a spectrogram, before turning to image processing tools. Harmonics produce variations of the same curve. By computing the dominant frequency, we can find that curve. But we are not just interested in one curve, or *ridge*, that forms part of a *syllable*, we're interested in multiple ridges, specifically three. Therefore, just computing the dominant frequency is not enough. For this reason, the true purpose of this first step is to find the dominant frequency at multiple *levels*. To do this, a dominant frequency is found; and then used to alter the spectrum of frequencies, by modifying the original spectrogram to *de-emphasize* the area where the dominant frequency was found. The new spectrogram is then used to find another dominant frequency. This process is repeated three times. The algorithm used to compute the dominant frequency is described in (Telgarsky, 2013).

Step 2 – Spectrogram Binarization In this section we restrict the color palette to two colours. In other words, we *binarize* the spectrogram plot. Unlike in previous sections where we binarized images using rather involved techniques, here we incorporate a much simpler strategy. We start by computing a threshold value, k, estimated for each data point. This value is based on the mean and standard deviation of the energies from a spectrogram, S, represented by pixels P. Henceforth, all points with a pixel colour under k are turned black, whilst the others are turned white i.e. $S_{P\leq k} \mapsto 0$, $S_{P>k} \mapsto 1$. Moreover, we utilize the $\underline{\text{original}}$ dominant frequency found in the previous step, to create vertical boundaries that are estimated to $\underline{\text{not}}$ contain any syllables. Consequently, the pixels of all the points in the spectrogram that are contained within these vertical boundaries are turned black.

Step 3 – Contour Extraction In this step, we employ the Marching Squares algorithm, (Maple, 2003), just as we did in previous sections, to find all the contours within the binarized spectrogram. We create horizontal boundaries based on the numerous dominant frequency curves, computed at different levels, in Step 1. We then filter for contours that include a point that lies within at least one of the horizontal boundaries.

Step 4 – Connecting Neighbouring Contours It is possible that there are contours found in the previous step that represent the same harmonic; this is an easy mistake, especially given the simplistic nature of the contour extraction and image binarization methods employed. We tackle this problem in this step, by connecting neighbouring contours, with the aim of obtaining full ridges. To do this, for each contour in a spectrogram, we search for neighbouring contours that are to the left or to the right of the contour. We then join the possible neighbours together into one longer curve. All curves are subsequently smoothed using a univariate spline[157], with degree parameter[158] $k = 1.5$, and are reparameterized to have a chosen number of points (the default is 100 points).

Step 5 – Segmenting Syllables The purpose of this final step is to group ridges together into syllables. We first compute thresholds based on the average x and y ranges across all ridges in the spectrogram in order to filter out contours with very different ranges. Next, we compute the average x coordinate and round it to the nearest multiple of 10. We then use these values to group contours with similar average x-coordinates, in order to segment the ridges into groups. We then remove all contours that are in groups of less than three. An example of the outcome of this step can be seen in Figure 5.15, where we see syllables containing three ridges on a spectrogram from a recording of a male kiwi.

[157]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html

[158]We note that the degree parameter was chosen experimentally after tests comparing connected ridges showed that with $k = 1.5$, the curves were sufficiently smoothed whilst their true shape was retained.

[158]https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html

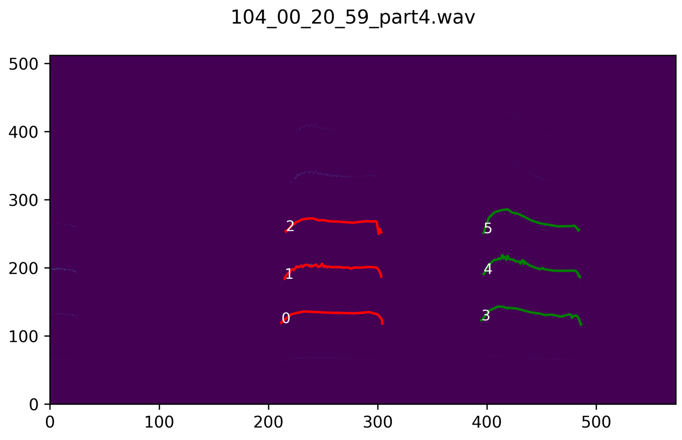


Figure 5.15: A binarized spectrogram from a 5-second kiwi call recording. Here, we see two syllables in red and green, each containing three ridges.

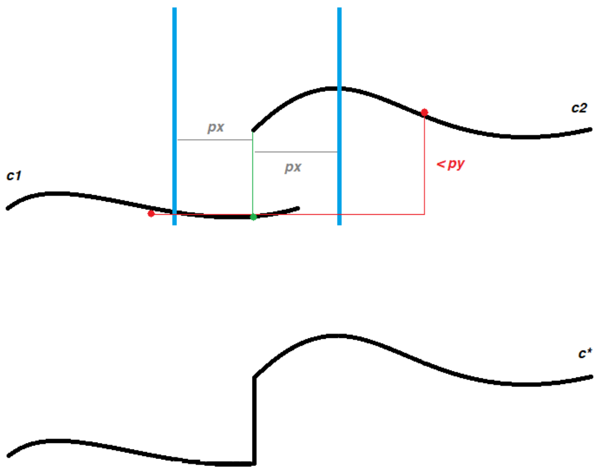


Figure 5.16: A diagram demonstrating Step 5 of Algorithm 15. Top: here we show an example of a contour c_2 being connected to a contour to its left, c_1. The blue lines demonstrate the boundary range for the end-point of c_1 and the red line measures the difference in the average y-values of c_1, c_2. The variables p_x, p_y are the boundary parameters described in Algorithm 15. Bottom: in this plot, we see the new, connected contour, made up of $c_1\big|_{x \le x_0}$ and c_2 where x_0 is the x-value of the first (left-most) point in c_2. Note that the straight line connecting the two segments is reparameterized in the subsequent smoothing step.

Algorithm 12: Multi-Level Dominant Frequency Extraction

`multi_dFreq(file,`k`=4,neighbours=5,`p`=0.5)`

Aim:

Computes a spectrogram and k dominant frequency curves.

Initial Step:

Load the file into a readable format, with some given sample rate.

Code:

1. Compute spectrogram S_0, using the Hann window function.
2. Compute the initial dominant frequency curve, d_0, from the spectrogram, S_0, using the algorithm described in (Telgarsky, 2013).
3. Find another $k-1$ dominant frequency curves:

```
for $i \in 1, \dots, k-1$ do
    Let $S_i := S_{i-1}$
    for $(x, y) \in d_{i-1}$ do
        • Create neighbourhood around point $(x, y)$ on $S_i$, based on the
          neighbours parameter.
        • Replace the intensities of the pixels in the neighbourhood with
          a series of random values between 0 and 1 (for example, by
          using numpy.random.rand).
    end
    Find dominant frequency curve, $d_i$, from $S_i$.
end
```

4. Split the initial dominant frequency curve, d_0, into multiple curves by estimating the start/end times of syllables. To perform this estimation, two threshold values t_1=`np.average(abs(`$(d_0)'_y$`))` and t_2=`np.average(`$(d_0)_y$`)-(p*np.std(`$(d_0)_y$`)` are computed. Points where the absolute value of the derivative falls below t_1, but y-value is greater than t_2 are deemed as syllable *end-points*. A list, D_x, containing the x-values of the end-point positions is created.

`return` $\{d_0, \cdots, d_{k-1}\}$ - k dominant frequency curves.

`return` S_0 `(numpy.array)` - original spectrogram.

`return` D_x `(list)` - estimated x-values of the syllable end-points.

Algorithm 13: Spectrogram Binarization

`binarizeSpec(spectrogram,`D_x`,`p`=0.625,nb=30,bound=15)`

Aim:

Creates a binarized version of a spectrogram plot.

Initial Step:

Compute `spectrogram` / syllable end-points, D_x, with Algorithm 12.

Code:

1. Compute threshold value k`=numpy.average(S)+p*(numpy.std(S))` where `S=spectrogram.flatten()`.
2. Make a copy, S^* of `spectrogram`, where every pixel, $p_i \leq k$ is turned black, and every pixel $p_i > k$ is turned white.
3. Create vertical segments between the start and end points of syllables. These segments are based on D_x, whilst their width is also based on the neighbour parameter u`=int(numpy.shape(`S^*`)[1]/nb)`. These vertical segments are hence grouped into a list of lists, V.
4. Recolour all points within the vertical segments, V, as black.
5. Since it is not uncommon to have noise present at the lowest levels of a spectrogram, we recolour points near the base as black, where the base boundary is determined with `bound`.

`return` S^* `(numpy.array) -` k binarized version of `spectrogram`.

`return` V - vertical segments between syllable start/end-points.

Algorithm 14: Syllable Contour Extraction

`contExtraction(binSpec,dFreqs,ms_level=0.8,bound=20)`

Aim:

Find all contours of syllables in a binarized spectrogram, `binSpec`.

Initial Step:

Get dominant frequencies (`dFreqs`)/`binSpec` with Algorithms 12/13.

Code:

1. Find all contours in binarized spectrogram, `binSpec`, using the Marching Squares algorithm. Note that the *level* parameter used in the Marching Squares algorithm is determined by `ms_level` [see Image Processing chapter for more details about this parameter].

2. Compute boundaries ranges based on each of the dominant frequency curves $\{d_1, \cdots, d_k\} \in$ `DFreqs`. For each d_i, the boundary range is $D_i = d_i \pm u_i\%$ where the percentage value, $u_i\%$, is determined by $u_i\%$=`min(bound-(i-1)*5,10)`%.

3. For each contour (x_i, y_i) found in the first step, we compute $n_1, \cdots, n_k$ where k is the total of dominant frequency curves in `dFreqs`, and n_j is the number of points within the contour (x_i, y_i), that lie in the boundary D_j:

```
if ($\exists n \in \{n_1, \cdots, n_k\}$ such that $n \geq 1$) and (length($x_i$)$> 5$)) then
    Add contour, $(x_i, y_i)$, to new contour set, $C$.
else
    Discard $(x_i, y_i)$.
end
```

`return` C - list of lists containing the filtered contours.

`return` $[D_1, \cdots, D_k]$ - boundaries for dominant frequency curves.

Algorithm 15: Connecting Neighbouring Contours

`connectingContours(C,nb=30,d=1.5,param=True,N=100,`p_x`=15,`p_y`=15)`

Aim:

Join contours that form distinct syllables.

Input:

Filtered list of ridge contours, `C`, from Algorithm 14.

Code:

1. For each contour $c_i \in$ `C`, we create a list of contours, Nb_i, that neighbour c_1 that could possibly form part of the same syllable. The neighbouring condition depends on a boundary range on the x-values based on `min(`$(c_i)_x$`) - nb` and `max(`$(c_i)_x$`) + nb`. The neighbouring contour list, Nb_i, is then filtered to contain neighbours that are entirely on the left of c_i or entirely on the right.

2. For a contour $c_i \in$ `C`, with `length(`Nb_i`)`> 0, Nb_i is filtered again to include a maximum of two neighbouring contours: the *closest* neighbour on the left, and/or the *closest* neighbour on the right. Henceforth, going from left to right, the neighbouring contours are combined to form one contour. `C`*, is then the updated contour list containing the new contours and those without neighbours.

3. Smooth new set of contours, using a univariate spline with degree `d`.

```
4. if reparam == True then
       Reparameterize the contours to have N points.
   end
```

5. Check all pairs (c_i, c_j) $\in$`C`*, to see if any pairs could be *neighbours*. This time, the neighbouring assumption is based on two criteria: one dependent on the x-values of contours, whilst the other is dependent on the y-values. The former criterion checks whether `max(`$(c_i)_x$`)`$\subset$ [`min(`$(c_j)_x$`)-`p_x,`min(`$(c_j)_x$`)+`p_x], if c_i is to the left of c_j, or `min(`$(c_i)_x$`)`$\subset$ [`max(`$(c_j)_x$`)-`p_x,`max(`$(c_j)_x$`)+`p_x] if it is to the right. The second criterion is based on the absolute difference between the average y-values, $\overline{(c)_y}$, i.e. $|\overline{(c_i)_y} - \overline{(c_j)_y}| < p_y$. If neighbouring, we join c_i, c_j by a straight line, as seen in the diagram in Figure 5.16.

...

```
6. if new neighbours were found in Step 5 then
   |    Repeat steps 3-4, and update C* accordingly
   end

return C* (list) - updated set of smoothed, reparameterized contours
describing syllables from spectrograms.
```

Algorithm 16: Segmenting Syllables

`segmentSyllables(C,`x_{ub}`=2,`x_{lb}`=0.5,`y_{ub}`=3,nb=30,n_ridges=3)`

Aim:

Segment ridges in a spectrogram into syllables.

Input:

Filtered and connected ridges, `C`, from a spectrogram with Algorithms 14, 15.

Code:

1. Compute the x-ranges, X, and y-ranges, Y, for all ridges in `C`.

2. Create x and y limits. We compute an upper and lower bound, `[UB`$_x$`,LB`$_x$`]`, based on the average x-ranges multiplied by x_{ub} and x_{lb} respectively. Furthermore we compute another upper bound, `UB`$_y$=$\bar{Y} \times y_{ub}$, where $\bar{Y}$ is the average y-range of the ridges.

3. Create a new set `C`*, and iterate through ridges: $\forall c_i \in$ `C`,

```
if ($X_i \geq$ LB$_x$) and ($X_i \leq$ UB$_x$) and ($Y_i \leq$ UB$_y$) then
    Include $c_i$ in C$^*$
end
```

4. For each ridge $c \in$ `C`*, compute the average x-coordinate, round it to the nearest multiple of 10 and append it to the list, $\bar{X}$.

5. $\forall$ ridges $c_i \in$ `C`*, create an array, N_i, of *neighbouring* (average x-)values, where N_i=`numpy.linspace(`$\bar{X}_i$`−nb,`$\bar{X}_i$`+nb,`k`)`, such that k=$(\frac{2}{10}\times$`nb`$) + 1$ and `nb` is a multiple of 10. $\forall c_i \in$ `C`*, we loop through all possible combinations to find contours in the same syllable, $\{c_i\}$:

```
if $\exists\ j$ such that $X_j \in N_i$ then
    $c_j$ and $c_i$ belong to the same syllable, $\{c_i\}$.
end
```

We then create a new set, `syllables`. $\forall\ i \in [0,$`length(C`*`)`$]$:

```
if length($\{c_i\}$)$\geq$n_ridges then
    $\{c_i\}$ is included in syllables.
end
```

and `syllables = numpy.unique(syllables)`.

`return syllables` - list of ridges, grouped by syllables.

5.2.5 Quantifying Differences between Kiwi Calls

Now that we have described the structure of kiwi calls and outlined how to extract syllables from spectrograms, we focus on how to analyse these acoustics. Our goal is to seek out differences between calls represented by curves on spectrograms, by focusing, primarily, on the shapes of the syllables. This leaves us with a few options on how we can proceed, such as pairwise distance computations or methods of dimension reduction.

Overall in this thesis, we have obtained successful shape classification results in previous projects seen in Chapters 4 and in Section 5.1, that employed methods of elastic shape analysis. Recall that the metrics used in these methods were invariant to shape-preserving transformations. Although this was useful in those projects, the translation invariance is not very helpful here, as we lose potentially vital frequency information. Henceforth, in this project, along with the shapes, we include various frequency information, in order to classify kiwi based on the syllables in their calls.

In our aim to classify individual kiwis by their calls, once again, we take on a hybrid approach involving elastic shape analysis and machine learning. Using a dataset of *trios* (i.e. syllables containing exactly three ridges), we incorporate a variety of methods to perform our classification; all of which will employ the SRVF framework. In the following sub-sections, we will outline our kiwi call project in more detail, describe the varying methods and algorithms we use, and present some classification results comparing each of the methods.

Dataset of Trios

In collaboration with AviaNZ, we obtained a dataset of approximately forty call recordings of the North Island Brown Kiwi (Apteryx Mantelli) taken from Ponui Island, New Zealand[159]. Our dataset featured four individual male kiwi, with a ground truth label of the caller.

[159]Though the name may suggest that the species are solely found on New Zealand's North Island, North Island Brown Kiwi are also found (and indeed have more stable populations) within off-shore islands around the North Island, such as Kapiti Island and Ponui Island. For more information on the North Island Brown Kiwi, and in particular, their calls and birdsong, we refer the reader to (Corfield et al., 2008).

Our first step in data processing is to segment the audio files into clips of no more than 5 seconds long. This can be achieved using any audio manipulation software or library, such as the audio manipulation Python library, pydub[160]. From here, we take on the 5-step process described by the five algorithms in the preceding subsection, in order to acquire a dataset of ridges, grouped into syllables, from spectrograms of recordings.

Using Algorithm 16 with the default parameters (namely, `n_ridges=3`), we obtain syllables consisting of three or more ridges. Although the code can be easily modified to restrict the number of ridges to an exact amount (for example by choosing the top three or the lowest three ridges), we opt for a more manual approach. We plot all ridges found after Algorithm 16 on spectrogram plots (such as the plot seen in Figure 5.15), and evaluate each by-eye. By manually evaluating the spectrogram plots, we can select three ridges to represent each syllable (for the minority that originally contain greater than three ridges), using our own judgement, and can also omit *poorly* represented syllables and ridges that our algorithms may have missed in the numerous filtering stages. Though this approach may seem tedious, due to the fairly small amount of original samples, it does not take too long; especially, as we have found that most syllables extracted using our methods, do only contain `n_ridges` (three) ridges. In future implementations the code will be updated to avoid this manual step. Subsequently, after the manual filtering steps, we obtain training and testing sets, consisting of 87 syllables (*trios*), thus, a total of $87 \times 3 = 261$ ridges. Note that the modest sample size is because this is a proof-of-concept trial.

In line with our ambition to include *frequency* information with our analysis, in addition to the coordinates of the three ridges in each syllable, our dataset also contains the following data for every ridge, where y-coordinates represent the frequency, and x-coordinates represent time:

Mode Frequency: To find the mode frequency, we round the y-coordinates of each ridge to the nearest integer and compute the mode average.

Median Frequency: We compute the median of the y-coordinates rounded to three decimal places (for *cost-saving* reasons) for each ridge.

[160] https://pypi.org/project/pydub/

Ridge Length: We compute the x-range of each ridge. Since the x-coordinates of our ridges are time-increasing, we define the segment length as simply the first x-value subtracted from the last. Once again, to save on computational costs, the values are rounded to 3 decimal places.

Combining Harmonics in the Classifications of Kiwi Calls

Throughout this thesis, the basis has always been the quantification of differences between two shapes. Here, things are a little different. Our aim remains similar, as we wish to define a way of quantifying differences between two kiwi calls. However, as each syllable is made up of three ridges, we are no longer dealing with two shapes, and instead, we have six.

There are multiple avenues we can explore from here, that will help us classify our collection of calls. We consider three of these avenues in particular:

1. Classification of ridges, without the grouping of syllables.
2. Weighted sum of differences between pairs of ridges in pairs of syllables.
3. Computation of differences of syllables in a product space.

The first approach is no doubt the simplest. Here, we quantify differences between the shapes of two ridges, without focusing on their group. Though this method generally does not rely on syllable information, there are ways in which we could incorporate certain syllable information, such as a ridge's position within a syllable (i.e. whether it's the lowest ridge, the centre ridge, or the highest ridge).

The second approach involves comparisons between pairs of ridges within two syllables. In other words, for two syllables, S, T, we quantify differences between pairs (s_i, t_j), $\forall\ s_i \in S,\ t_j \in T$, where $1 \leq i, j \leq 3$. This *comparison* could be a distance computation between the shape of the ridges s_i and t_j and/or the differences between their *frequency information* (for example, their segment length). Hereafter, the option remains to compare all nine pairs of ridges, to focus on the three same-level comparisons, or some alternative combination. The final comparisons can then be combined into one value, quantifying the differences between the two calls.

Lastly, we consider the avenue exploring the quantification of differences between syllables, S, T, in a product space defined by their ridges. We can define an open ridge, $s \in S$, as the mapping $s : I \mapsto \mathbb{R}^2$ for some interval I. Thus, a syllable can be thought of as the product of three ridges, $\{s_1, s_2, s_3\} \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$. The aim of this approach is to define a metric that computes a distance between the shapes of S and T on the product space; with the intention that the metric is then incorporated into a desired machine learning classifier, to classify the kiwi calls. As this approach has not yet been finalised, we discuss this avenue in the *future work* section of this project.

Next, we will go into more details about the three aforementioned approaches. In particular, we focus on the first two methods, which we developed much further. We discuss the methods of elastic shape analysis used to quantify differences, and outline the machine learning algorithms we tested and incorporated.

Support Vector Machine Classification

Our first method is aimed at classifying kiwi based on the ridges found in their calls. In this approach, we incorporate a dataset of ridge information that includes:

- x, y-coordinates of ridge,
- ridge *length* (range in the x-axis),
- mode frequency of ridge,
- median frequency,
- position in call (either 1-*lowest*, 2, or 3-*highest*).

Thus far in this thesis, we have discussed several classification projects, from classifying mussels to clustering Greek vases. In all of these projects, our analysis has been solely based on the shapes of the objects in our dataset. For this reason, we dealt with distance matrices, which led us to incorporate a KNN classifier for the classification step of each of those projects. As mentioned earlier, one important place where this project differs from the others, is in its requirement for the inclusion of additional information, as well as the differences in shapes. Therefore, for N samples in our dataset (where $N = 88$, represents the number of individual calls), unless we plan to sum the *differences* between the samples (as we will do so in the second approach, which we discuss later), we are no longer dealing with an $N \times N$ distance matrix. Instead, for each of the N samples, we possess a large array of information (as seen in the list above), leading to a very high-dimensional dataset. Consequently, we now turn to a different machine learning classifier to try: support vector machines.

Support vector machines, SVMs, are supervised learning methods, predominantly developed in the 1990s, by Corinna Cortes and Vladimir Vapnik, (Cortes and Vapnik, 1995), based on methodology first described in the 1960s by Vapnik and the mathematician Alexey Chervonenkis. With their robustness, and ability to work well with high-dimensional data, whilst not being too computationally expensive, SVMs remain one of the most widely-used machine learning classification algorithms.

Broadly speaking, the objective of SVM classification is to find optimal decision boundaries (or *boundary*, in binary classification) that best separate the data into the given classes. Whilst the dimensions of the boundaries are determined by the number of *features*[161], the *type* of boundary can vary as desired, with the simplest example taking a linear form[162], with the aim to find an optimal, $(N-1)$-dimensional hyperplane in $\mathbb{R}^N$, where N is the number of features in the dataset. Here, the algorithm searches for the hyperplane that best separates out the data by their classes, where the *optimality* of the decision boundary is based on a misclassification parameter (often labelled as C), and, more significantly, on the maximisation of a *margin* between the hyperplane and the data-points of each of the classes. More specifically, the margin is measured by the distance between the decision boundary to points in each class that are closest to this decision boundary, i.e., the *support vectors*. Finally, new data-points are classified based on the sides of the decision boundaries they lie within, when plotted.

The SVM transforms non-linearly-separable data-points into a higher-dimensional space, where the data-points can be linearly separated. As such transformations may be computationally expensive, the process can be side-stepped, by defining a metric on the new, higher-dimensional space, using a *kernel function*. These functions can vary and be *customised*, allowing SVMs to be versatile classification algorithms. Here, we consider the standard linear approach, as well as the commonly used Radial Basis Function, RBF,

[161]I.e. the number of categories or *information* we have of each sample, in a dataset.

[162]We note that the decision boundary does not need to be linear. Other examples include polynomial curves.

kernel for our SVM classifiers. The RBF kernel can be described with:

$$K(v_1, v_2) = \exp -\gamma ||v_1 - v_2||^2_{\mathbb{L}^2} \tag{5.15}$$

where v_1, v_2 are data-points[163], and the parameter, $\gamma > 0$, is a scalar determining the with of the kernel.

For further information regarding support vector machines and the mathematics behind kernel functions in particular, we recommend (Bishop, 2006). We now move onto the topic of how SVMs can be combined with elastic shape analysis in order to classify kiwi calls.

Method 1 – SVMs & Karcher Mean Analysis

In this first method, we are left with a rather large dataset, with a very high number of features, due to the x, y-coordinates of each ridge (represented by 100 equally-spaced points) being included. This is problematic for the classifier; not only due to its size and hence possibility of over-fitting and expensive computations, but because the x-coordinates and the y-coordinates will be treated independently, as individual features, which is not ideal. To overcome this, we transform the shape of the ridges into a handful of values, such as *principal components*, computed using tangent PCA (tPCA), which can then replace the original coordinates, before applying the SVM classifier. We note that for a more concise overview of tPCA, we refer the reader to Section 2.5.3 of the background chapter, and to Chapter 7 of (Srivastava and Klassen, 2016).

In this project, for two open curves $c_1, c_2 : I \mapsto \mathbb{R}^2$, the elastic distance between the curves, $d(c_1, c_2)$, involves a diffeomorphic mapping to *match* c_1 to c_2, within the square root velocity (function), SRVF, framework; using a dynamic programming algorithm to perform the computations. By incorporating this method and framework, we compute distances between the <u>shapes</u> of our kiwi ridges, without any linearity approximations that traditional methods[164] often take. This leads to Karcher mean computations that are more representative of the shapes in the dataset, and hence, more indicative principal components. In Chapter 4, we saw plots that visualised and emphasised this point (for

[163]Without loss of generality, v_1 can be thought of as a labelled data-point, whilst v_2 can be considered a new data-point, waiting to be classified.

[164]We refer to methods commonly used for analysing real-world shape data, such as Morphometric methods.

example, see the shapes of Ancient Greek vases in Figure 4.15 and Figure 4.16). For more details regarding Karcher PCA and distance computations within the SRVF framework, we refer the reader to the Background chapter of this thesis.

In order to classify male kiwi by their calls, we apply an SVM classifier to a dataset of ridge information, where the ridge coordinates are replaced with N principal components, as discussed. We test out three kernels for our SVM classifier: linear, RBF, and polynomial. Though they all require the same misclassification penalty parameter, C, the RBF also requires the variable γ (see previous subsection), whilst the polynomial kernel requires a degree parameter, δ. As the choice of parameter can have a significant impact on the classification results, we optimize each of the parameters using a cross-validation search built within the scikit-learn[165] library. The approach involves an exhaustive grid-search based on initial values we set. Our initial values ranged from an array of uniform random variables (using the function `scipy.stats.uniform()`, from the scipy[166] library), as well as manually selected values, where $0.5 \leq C \leq 150$, $1e-7 \leq \gamma \leq 1e-1$, and $1.5 \leq \delta \leq 4$. From here, the method searches for *best* results, where we opt for the success to be measured with an F_1-score:

$$F_{\beta=1}(P,R) = (1+\beta^2) \times \frac{PR}{P(\beta^2)+R} = 2\frac{PR}{P+R} \tag{5.16}$$

where P, R are the precision and recall scores, as seen in Chapter 4.

During the parameter-optimization step, we fix the number of principal components, N, prior to further tests being carried out to analyse the affects of the number of principal components. We set $N = 20$ as this sufficiently covered the variability in the data whilst not being too computationally heavy for the optimization step. The training and test sets contain 102 and 159 ridges respectively, based on a quasi-random sampling, as done in previous projects. In order to analyse how the number of principal components can affect the results, we ran an SVM classifier on the test set, using the optimal parameters found in the earlier step, with $N \in \{3, 5, 15, 20\}$. The F_1-scores from our top two methods (the linear kernel and RBF kernel) can be seen in Table 5.1:

[165] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[166] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.uniform.html

SVM Kernel / N	3	5	10	15	20
Linear	0.793	0.792	0.800	0.787	0.767
RBF	0.678	0.719	0.719	0.719	0.713

Table 5.1: F_1-scores from SVM classification on kiwi ridges, with varying principal components (N), alongside additional frequency and call information.

Interestingly, Table 5.1 shows that the results do not vary too much as N is altered, though in both the linear and RBF kernel, a peak in F_1-scores can be seen in the middle mark, particularly at $N = 10$. The table tells us that the linear kernel for a SVM classifier, came out on top, with a maximum F_1-score of 0.8. However, recall that our global aim is to classify kiwi calls, and not specifically ridges. Therefore, in order to obtain a score on the classification of calls, we compute the mode[167] predicted class for each syllable. By employing such tactic, we find that the F_1-scores improve, with the top scores being 0.821 and 0.767, for the Linear and RBF kernel respectively. Though these scores are not too bad, in the next section, we will see how a different approach in classifying kiwi calls can result in even better scores.

Method 2 – KNN, Elastic Distances, & Optimization

Previously, in projects such as Mussel Identification in Chapter 4, we implemented a KNN classifier with a sole required input of a distance matrix. Once again, we return to this KNN algorithm (as described in Algorithm 9), and its bootstrapping-equivalent (Algorithm 10), to identify kiwi based on their calls.

In this approach, our main focus when comparing two calls is on the differences between the shapes of syllables, as well as an additional interest on the frequency of syllables. As there are six ridges between two syllables, our first step is to decide how to *define* these differences between the two syllables, via their ridges. In general, we have seen that the stronger ridges are closer to the fundamental frequency, with strength slowly diminishing with every increasing harmonic. Therefore, when comparing two syllables, it is possible that the lowest ridges are more alike, as are the next two and so on. For this reason, in this project, we opt for a *same-level*[168] comparison between ridges. In other words, the

[167]Note that, if multiple mode values are found, the predicted class of the lowest ridge is used.

[168]For the remainder of this section, we use the term *same-level* to refer to ridges that have the same

ridges will be sorted from lowest to highest, and direct comparisons will be computed between the two lowest ridges, the middle ridges, and the highest ridges. These ridge comparisons will then be part of a weighted sum that we use to define the differences between two syllables.

Once again, to quantify differences between shapes of ridges and hence syllables, we return to the method that resulted in the top classification scores (see Section 4.1.9): namely, elastic distances in the SRVF framework. As we are dealing with open curves in this project, we employ a dynamic programming algorithm, as outlined in Algorithm 1. Recall that this method compares the shapes between two open curves by minimizing the distance between the orbits of the square root velocity functions of the curves, as discussed in 2.4.4. Hence we use this distance metric to compare the distances between the shapes of our ridges.

Pairwise distances are computed between all same-level curves. Henceforth, for a dataset containing N syllables in *trios*, we obtain three $N \times N$ distance matrices, one for each *level*. Similarly, in order to compare the differences in *frequency information*, we compute the absolute differences between same-level syllables, which results in three *difference matrices* per variable (i.e. for mode frequency, median frequency, and segment length). Thus in total, we have twelve $N \times N$ matrices, comparing kiwi calls.

Our KNN classifier requires only one distance matrix, therefore, we need to find a way to merge our twelve matrices into one. This can be done using a weighted sum. However, optimizing twelve parameters would be very computationally expensive, almost regardless of what optimization method is used. In lieu of this, we test out three options[169]:

Option 1: We focus solely on the three <u>distance</u> matrices, D_1, D_2, D_3, (i.e. the distances between the lowest level ridges, the middle-level ridges, and the highest level ridges, respectively) and form one matrix, D, after optimizing for the parameters $w_1, w_2, w_3 \in \mathbb{R}$.

$$D_{\text{opt1}} = \omega_1 D_1 + \omega_2 D_2 + \omega_3 D_3 \tag{5.17}$$

position within their respective syllables, e.g. two ridges that are both the lowest in their *trio*, can be described as being *same-level*.

[169]An additional, albeit *unsuccessful*, approach involved a version of Option 2, that excluded the median frequency difference matrices.

Option 2: Use Option 1 to find optimal parameters for D_{opt1}. Compute matrix based on D_{opt1} and a weighted sum with scalars $\alpha_i, \beta_j, \gamma_k \in \mathbb{R}$, $1 \leq i, j, k \leq 3$, for the *difference matrices*, D^{mode}, D^{med}, D^{len}, representing the mode / median frequency, and segment length differences respectively.

$$D_{\text{opt2}} = D_{\text{opt1}} + \sum_{i=1}^{3} \alpha_i D_i^{\text{mode}} + \sum_{i=1}^{3} \beta_i D_i^{\text{med}} + \sum_{i=1}^{3} \gamma_i D_i^{\text{len}} \tag{5.18}$$

Option 3: We focus on all three distance matrices and <u>one</u> *difference matrix*. We test a basic optimization on all matrices D_i^X where $X \in \{\text{mode}, \text{med}, \text{len}\}$ and $i \in \{1, 2, 3\}$, to find the *best*-performing *difference matrix*, D^*. Once decided, we optimize the scalars $\omega_1, \omega_2, \omega_3, \omega_4 \in \mathbb{R}$:

$$D_{\text{opt3}} = \omega_1 D_1 + \omega_2 D_2 + \omega_3 D_3 + \omega_4 D^* \tag{5.19}$$

The motivation behind choosing these three options was so that we can learn more about which ridge level and additional information was more vital when it comes to the classification of syllables. As all three options involve weights, by optimizing these weights, we can learn a lot of about the data. A further motivation was due to the relative mathematical simplicity and thus computational ease of such methods.

In order to optimize the parameters in the weighted sums presented across all three options, we employ a *naive* grid-search approach on a KNN classifier. A traditional grid-search approach (such as scikit-learn's cross validation method used in the previous section) begins with an initial selection of possible values for each parameter, followed by an evaluation, which in our case involves the implementation of a KNN classifier on the distance matrix created using the chosen parameters, and an F_1-score computed from the results. The parameters that resulted in the highest F_1-score are hence deemed as the *optimal parameters*. Where our approach differs from the default algorithms is that we opt for a more *manual* approach. Here, a grid-search is applied multiple times, where the parameter options become *denser* with each grid-search iteration. Each grid-search allows us to learn more about the parameters and their relationships with each other, enabling us to choose the next set of parameters tactically based on the results of previous iterations, and to also include possible conditions e.g. for two arbitrary scalar parameters α, β, we can insist that $\alpha \leq \beta$.

As an example, we outline a grid-search method in Algorithm 17, aimed at finding the

Algorithm 17: Grid-Search Parameter Optimisation Example

```
optimiseDistMat(D,trainTest,I,$W_1$,$W_2$,$W_3$,k=np.linspace(3,6,4))
```

Aim:

Optimise parameters $\omega_1, \omega_2, \omega_3 \in \mathbb{R}$ in Equation (5.17). Find the best parameters from the options $\omega_i \in W_i$ for $i = \{1, 2, 3\}$, where the distance matrices used in the weighted sum make up the variable `D (np.array)`, i.e. D_1=`D[0]`, D_2=`D[1]` and D_3=`D[2]`.

Initial Step:

From the index data (containing the name and true classification of calls in the sample), `I`, create N training / testing sets, `trainTest`.

Code:

1. Create parameters `all_scores==[]`, `top_F=0`, `top_params=[0,0,0]`.

2. Start grid-search process, with a classifier, `kNN`, (see Algorithm 9).

```
for $\omega_1 \in W_1$ do
    for $\omega_2 \in W_2$ do
        for $\omega_3 \in W_3$ do
            if $\omega_1 \geq \omega_2$ then
                $D_{\text{opt1}}$=$\omega_1$D[0]+$\omega_2$D[1]+$\omega_3$D[2]
                scores = []
                for sample $\in$ testTrain do
                    F=kNN($D_{\text{opt1}}$,sample,idxData=I,neighbours=k)
                    scores.append(F)
                end
                avg_F=np.average(scores)
                all_scores.append(avg_F)
                if avg_F $\geq$ top_F then
                    top_F=avg_F
                    top_params=[$\omega_1$,$\omega_2$,$\omega_3$]
                end
            end
        end
    end
end
```

`return all_scores` - average F_1-scores for all parameters.

`return top_params` - optimal parameters.

optimal parameters $\omega_1, \omega_2, \omega_3$ in Equation (5.17) for Option 1. We test this algorithm using lists of uniformly distributed values, W_1, W_2, W_3, in order to find the *optimal* parameters. Moreover, in this example, we add an extra condition, stating that $\omega_1 \geq \omega_2$, as previous results showed that F_1-scores were higher when this condition was met.

By taking on a more *manual* approach to parameter optimization, we gather a better understanding of the parameters involved, which enables us to improve the initial parameter estimation values in a grid-search algorithm, as well as the algorithm itself, and hence, the results. For instance, to test Option 1, we begin by employing the same initial values across all parameters. In turn, this allows us to make meaningful comparisons of results between parameters, such as the assumption incorporated in Algorithm 17: that results improve when $\omega_1 \geq \omega_2$. This is illustrated in Figure 5.17, which shows a heat map plot of the average F_1-scores for variables ω_1, ω_2, after a basic, initial, grid-search optimization, with values between 0 and 1. Similar tests simultaneously revealed another interesting result, which is that F_1-scores are higher when ω_3 takes on smaller values whilst ω_1, ω_2 take on the larger values, as shown in Figure 5.18. By learning this information, parameter conditions and initial values can be updated accordingly, in further iterations of the grid-search process.

We employ a *manual* grid-search approach to optimize the necessary parameters in all three options (i.e. Equations (5.17), (5.18), and (5.19)). In total, 10 varying training and testing sets were made in an attempt to reduce the influences of the training / testing splits[170]. For Option 1, we exclusively use all ten training and testing sets in the grid-search process (which refers to the parameter `trainTest` in Algorithm 17). Since Option 2 and Option 3 contain more parameters, to speed up the computations, a subset of two and five training and testing sets, respectively, are used instead.

Out of the three options, the simplest, and computationally least-expensive approach (in regards to parameter optimization) is undoubtedly Option 1. Here, a grid-search is enabled, in order to find three optimal parameters $\omega_1, \omega_2, \omega_3 \in \mathbb{R}$ in Equation (5.17). Meanwhile, Equation 5.18 for Option 2 holds the most parameters needed to optimize, namely the three parameters for D_{opt1}, as well as the nine scalars needed to sum each of the *difference matrices*. Finally, as seen in equation (5.19), Option 3 requires only four

[170]Note that in subsection 5.2.5, we discuss the reasons for these influences in more detail.

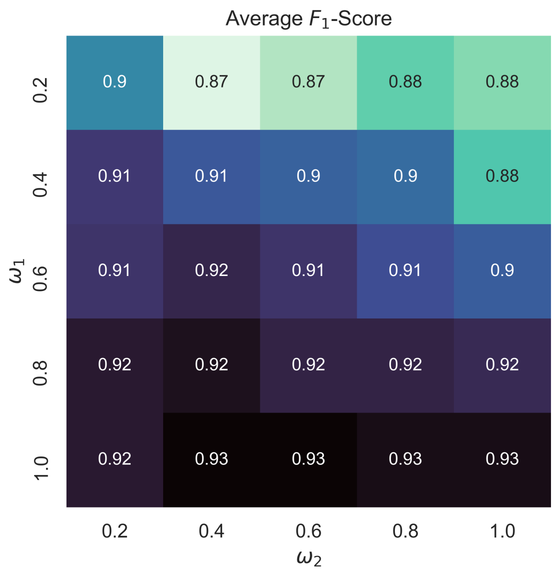


Figure 5.17: A plot showing the average F_1-scores for parameters ω_1, ω_2, resulting from a kNN-based, grid-search, to optimize parameters $\omega_1, \omega_2, \omega_3$ in D_{opt1} (see Equation (5.17)). The axes represent the list of values tested in the grid-search algorithm (Algorithm 17). The darker the square, the higher the average F_1-score. This plot shows that the average F_1-scores are greater when $\omega_1 \geq \omega_2$.

scalars, thus, computationally, it may not seem too different to Option 1. However, there is the added, albeit rather hidden, step of choosing the *difference matrix* D^*, which can be done in a variety of ways. In this project, to decide on the matrix, D^*, we run a grid-search optimization algorithm, similar to Algorithm 17, but catered for four matrices. Here, the variable `D` contains the three distance matrices D_1, D_2, D_3, as well as an added *difference matrix*, D^*. The algorithm also includes an extra variable W_4 to find the optimal scalar for D^* (corresponding to ω_4 in Equation (5.19). This grid-search algorithm is run with all *difference matrices*, i.e. for $D^* = D_i^X$ where, $X \in \{\text{mode}, \text{med}, \text{len}\}$ and $i \in \{1, 2, 3\}$. The *difference matrix* resulting in the highest top-average F_1-score is then chosen as the final matrix, D^*, and focused-on in further grid-search optimization tests for Option 3. The results from all nine tests can be seen in Table 5.2, which shows that D_2^{med} (i.e. the differences between the median frequencies of the middle-level ridges in kiwi calls) corresponds with the highest F_1-scores.

Overall, the classification results obtained alongside the optimal parameters during the grid-search processes prove to be rather promising, in particular, for Option 1 and Option

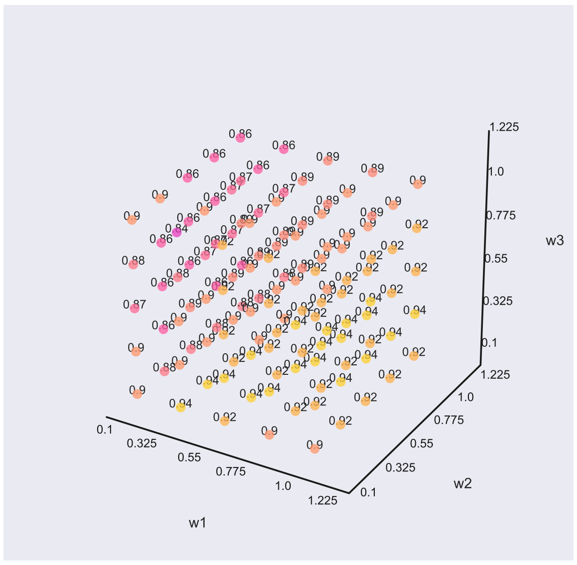


Figure 5.18: F_1-scores resulting from a grid-search using a kNN classifier to optimize parameters $\omega_1, \omega_2, \omega_3$ in Equation (5.17). In this example, all test values were equally uniformly distributed, with W_1, W_2, W_3=`np.linspace(0.1,1,5)`. The colour of the markers represent the F_1-scores, where the more yellow the colour is, the higher the score. The yellower markers at the bottom shows us that F_1-scores are higher when ω_3 decreases.

3, where the top average F_1-score is consistently above 0.8, and the maximum F_1-scores are above 0.9. On the other hand, the top average F_1-score obtained from the initial grid-search for Option 2, is around 0.7. This poor performance can be attributed to the low number of training and testing samples, but more prominently, to the lower number and *wider* set of options in the initial value lists for the grid-search, which are purposely selected to offset the high number of parameters required for the grid-search optimization[171]. Although a second grid-search test was done for Option 2, this time with a denser list of initial parameter values, the top average F_1-score still remains around 0.7. For this reason, we decided to not pursue this option further.

After numerous grid-search optimization processes, the optimal weighted sums of matrices

[171]As high numbers of parameters would significantly increase computation rates in a grid-search process, we look at 3 possible values, initially, for each parameter, when optimizing for Option 2. In comparison, for Option 1 and Option 3, the number of parameter in the initial grid-search lists are 5 and 4 respectively. This number is then increased (and the lists were made *denser*) accordingly in further iterations.

D^*	F_1-Score
D_1^{mode}	0.824
D_2^{mode}	0.826
D_3^{mode}	0.804
D_1^{med}	0.870
D_2^{med}	0.876
D_3^{med}	0.806
D_1^{len}	0.804
D_2^{len}	0.767
D_3^{len}	0.707

Table 5.2: Maximum (average) F_1-scores resulting from a grid-search parameter optimization for $\omega_1, \omega_2, \omega_3, \omega_4$ in D_{opt3} (see Equation (5.19)), with differing D^*. In this example, all four parameters in all nine grid-search experiments (as there are three *difference matrices* at three levels) use the same, uniformly-distributed, test parameters, namely, `np.linspace(0.1,1,4)`. Our results show that the highest score, (as highlighted in red), is obtained when $D^* = D_2^{\text{med}}$.

for Option 1 and Option 3 were as follows:

$$D^*_{\text{opt1}} = 0.8D_1 + 0.45D_2 + 0.3D_3 \tag{5.20}$$

$$D^*_{\text{opt3}} = 0.85D_1 + 0.5D_2 + 0.5D_3 + 0.02D^* \tag{5.21}$$

for distance matrices D_1, D_2, D_3, containing the distances between the shapes of *same-level* ridges from kiwi calls, in SRVF space, and the *difference matrix*, $D^* = D_2^{\text{med}}$, containing the absolute differences between the median frequency of the middle-level ridges.

Interestingly, Equations (5.20) and (5.21) show us that grid-search optimization incorporating both Option 1 and Option 3 result in similar *optimal* parameters being found for the scalars corresponding to the weights of the distance matrices. Although, we remark that the scalar attributed with the matrix D_3 (i.e. the distances between the top-level ridges, which, as emphasised previously, tend to be the least prominent of the three harmonics that make up the syllables in our dataset) is weighed noticeably more heavily in D^*_{opt3}. Another interesting, and rather surprising outcome is the small scalar attributed with D^*. This tells us that the additional frequency information is not regarded as important as the distances between the ridges when classifying kiwi calls.

Finally, kNN classification is performed on the matrices $D^*_{\text{opt1}}, D^*_{\text{opt3}}$ using 10 varying training and testing sets; the results of which can be seen in Table 5.3. The success rate in both classification tests are rather remarkable, with the highest F_1-scores equalling 0.936, and the average being 0.875 and 0.885 for $D^*_{\text{opt1}}, D^*_{\text{opt3}}$ respectively. Intriguingly, despite the fact that scalar used for D^* in D^*_{opt3} was so small it could seem disposable, we find that the overall classification results are slightly better for D^*_{opt3}, than they are for D^*_{opt1}. This tells us that the added frequency information, though small, can make a difference. Importantly, these results demonstrate that incorporating a kNN classifier with elastic distances (as well as median frequency differences) between ridges in kiwi syllables, can be a highly successful technique in identifying individual kiwi.

Matrix	Max F_1-Score	Average F_1-Score	Standard Deviation
D^*_{opt1}	0.936	0.875	0.042
D^*_{opt3}	0.936	0.885	0.038

Table 5.3: F_1-scores from kNN classification on kiwi ridges. Here, the classifier is implemented on optimised matrices, D^*_{opt1}, D^*_{opt3}, which incorporate Option 1 and Option 3 respectively. In total, classification tests are performed on 10 training and testing sets, with the number of neighbours `k=np.linspace(3,6,4)`. The table shows the maximum F_1-score out of the 10 tests, as well as the mean average and the standard deviation of the F_1-scores.

A Note on Robustness

Recall that our work here is a pilot test on how we can utilize shape analysis to recognise kiwi calls. Therefore, we were limited in the amount of data we could analyse, leading to the small size of our sample and its very unequal proportions of individual kiwis. The size and the proportions of data can have a great influence on classification. Another affect on classification could be due to a stronger similarity between the syllables of some birds, than initially suspected. Thereby, we believe that the variation in classification results depending on the training and testing split is a direct consequence of the dataset size, and the syllable similarities, and not necessarily down to a lack of robustness in our methods. In general, sound data from the wild can be rather noisy and whatever data extraction technique employed, whether it's syllable extraction from spectrograms or not, is not inherently simple. Thus for the classes (i.e., individual kiwi) that only had a small handful of audio segments, it is no surprise that the classifier often failed to match ridges. This is why studies in the field of audio classification often employ large datasets to increase the accuracy of their results, such as in (Ovaskainen et al., 2018), where the authors used a dataset of almost 200,000 audio segments to create an animal sound identifier. In order to truly examine the robustness of our methods, a larger dataset is certainly required. This is something planned for the future of this project.

5.2.6 Summary & Future Work

In this project, we have studied an assortment of methods that can be incorporated, in order to identify individual kiwi from the syllables of their calls. Overall, these methods have had varying success, from a top F_1-score of 0.821 using an SVM with a linear kernel, to a top F_1-score of 0.936 with a kNN approach; where all results use the same series of training and testing sets. In lieu of the promising results we have obtained thus far, there is now more work to be done, following on from our experiments.

Method 3 – Product Space Matching

The final method we introduced earlier, on quantifying differences between kiwi calls, has yet to be finalised. Nonetheless, we outline the ideas and the foundations of what can someday be used to analyse kiwi calls, by the *product* of their ridges.

In this project, kiwi syllables are defined by a triple of ridges. Thus, we consider two syllables c_1, c_2, such that $c_1 = \{s_1^{\mathbf{1}}, s_2^{\mathbf{1}}, s_3^{\mathbf{1}}\}$, $c_2 = \{s_1^{\mathbf{2}}, s_2^{\mathbf{2}}, s_3^{\mathbf{2}}\}$ where $s_i^{\mathbf{1}}, s_i^{\mathbf{2}} : I \mapsto \mathbb{R}^2$, $\forall i \in \{1, 2, 3\}$ for some interval, I. Our aim is to find a way to optimally match the syllable c_1 with c_2. As studied in the Background chapter, recall that the general goal of an *optimal matching*, for example between two ridges $s_1^{\mathbf{1}}, s_1^{\mathbf{2}}$, is to construct a smooth function ϕ, such that $(\phi, s_1^{\mathbf{1}}) = s_1^{\mathbf{2}}$, where $(\phi, \cdot)$ is the action of composition (see section 2.4.2). More specifically, $\phi \in \mathrm{Diff}(\Omega)$ is a diffeormorphism that *morphs* the shapes of our curves; where in the case of deformations on ridges, $\Omega = \mathbb{R}^2$. But the question remains, *how can such optimal diffeomorphism be defined on a set of ridges, i.e. on a syllable?* We consider two possibilities:

1. For two calls, c_1, c_2, we search for <u>one</u> diffeomorphism ϕ that acts on all three ridges in the same way. To do this, we define the calls, c_1, c_2 as a product of their respective

ridges, such that $c_1, c_2 : I \mapsto \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$. As $\mathbb{R}^2$ is a smooth manifold, the product space $\Omega = \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$ is also smooth. Thus, we can search for a smooth function $\phi \in \text{Diff}(\Omega)$, such that (ϕ, c_1) and c_2 are aligned, where:

$$(\phi, c_1) = \{s_1^{\mathbf{1}} \circ \phi, s_2^{\mathbf{1}} \circ \phi, s_3^{\mathbf{1}} \circ \phi\} \tag{5.22}$$

From here, one example of how we can find an *optimal* diffeomorphism, ϕ^*, is by introducing a minimization problem as follows:

$$\phi^* = \arg\min_{\phi} \sum_{i}^{3} d((\phi, s_i^{\mathbf{1}}), s_i^{\mathbf{2}}) \tag{5.23}$$

for ridges $s_i^{\mathbf{1}} \in c_1$, $s_i^{\mathbf{2}} \in c_2$, and some distance metric, d. In other words, to match two syllables, c_1, c_2, we search for a diffemorphism that *optimally* deforms the curves in c_1, by minimizing the sum of the pairwise distances between deformed curves in c_1 with those in c_2.

2. Alternatively, we can consider an optimal diffeormophism in conjunction with individual diffeormorphisms between same-levels ridges, in two syllables c_1, c_2. Our space is defined as a product of smooth manifolds, namely $\Omega = \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$. We then search for diffeomorphisms ϕ_i that align $s_i^{\mathbf{1}}$ with $s_i^{\mathbf{2}}$ for all $i \in \{1, 2, 3\}$. Hence we define ϕ as:

$$\phi = (\phi_1, \phi_2, \phi_3) \tag{5.24}$$

To show that ϕ is also a diffeomorphism, we consider the following:.

Proposition 5.1 (Proposition 3.33 in (Lee, 2010)) *A product of continuous maps is continuous, and a product of homeomorphisms*[172] *is a homeomorphism.*[173]

This tells us that at the very minimum, ϕ a homeomorphism. However, for ϕ to be a diffeomorphism, we require the derivative of the map to also be continuous, to maintain smoothness. If we define the derivative of ϕ' on the product of the tangents spaces to our smooth manifolds, such that $\phi' = (\phi_1', \phi_2', \phi_3')$, we find that ϕ' is continuous as each ϕ_i' $\forall i \in \{1, 2, 3\}$ are continuous. Therefore, the function, ϕ is smooth. In a similar fashion, it can be shown that the inverse $\phi^{-1} = (\phi_1^{-1}, \phi_2^{-1}, \phi_3^{-1})$ is also smooth. Hence ϕ is a diffeomorphism.

[172]A homeomorphism is continuous function with a continuous inverse. All diffeomorphisms are homeomorphisms, but the converse is not necessarily true.

[173]For a proof of this proposition, we refer the reader to (Lee, 2010).

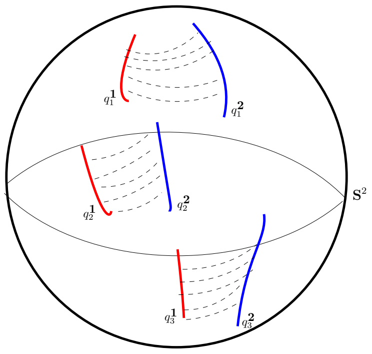


Figure 5.19: Illustration of geodesic paths to analyse two kiwi syllables, c_1 (red) and c_2 (blue). In this example, we employ the SRVF framework, in order to simplify geodesic computations. Recall that this is due to the space of shapes becoming the unit sphere, $\mathbb{S}^2$, where geodesics can be comfortably described with great circles. Here, our syllables are described as $c_1 = \{s_1^{\mathbf{1}}, s_2^{\mathbf{1}}, s_3^{\mathbf{1}}\}$, $c_2 = \{s_1^{\mathbf{2}}, s_2^{\mathbf{2}}, s_3^{\mathbf{2}}\}$, and the SRVFs are represented by $q_i^{\mathbf{1}}, q_i^{\mathbf{2}}$ for $i \in \{1, 2, 3\}$. By taking on the second approach, we perform an *optimal matching* by aligning the three same-level ridges within the two syllables. Note that the dotted lines represent the shortest arcs of great circles between points on the curves $q_i^{\mathbf{1}}$ and those on $q_i^{\mathbf{2}}$.

Conceptually, the task of such diffeomorphism, ϕ, can be thought of as the least *energy*-consuming way of transforming two shapes. As seen in the Background chapter, this transformation can be done with geodesics. Furthemore, by finding a geodesic path between two shapes, we can describe the energy required to transform, or *deform*, one shape into another, by computing the squared length of the geodesics. This, in turn, can be used to quantify differences between shapes, in our case, of kiwi calls.

From this point, it remains to formally define a metric on the product space, which can be used to quantify differences between two syllables. Though we note that this metric will be guided by the work done in (Holm et al., 1998). Once this metric has been defined, we will compute pairwise distances between syllables, and hence we can implement a machine learning classifier. This classifier will most likely be a k-NN classifier, as we have learnt, throughout this thesis, that a k-NN classifier works impressively well when implemented on a matrix containing pairwise elastic distances between shapes.

Additional Future Work

Further Species: Throughout this project, our focus has been on male kiwis, as it is known that their calls produce distinct curves (harmonics) when plotted on spectrograms. However, our global here is to expand our work to other species of birds. Though we must be careful, as we can certainly face some difficulties when branching out. For example, though male kiwis produce calls that can be extracted via the processing tools we highlighted in Section 5.2.4, the same cannot be said for the shrieking calls from female kiwi. However, though we may not see distinct curves, *shapes* do appear on their spectrograms, nonetheless. When testing contour extraction methods on a set of various female kiwi calls, we found that their spectrograms often exhibit rather *blob-like* shapes. In order to analyse these shapes, we extract closed curves that surround them. This extraction is done using similar methods described in this section. We start by finding the dominant frequency curve. In converse to the extraction of male kiwi calls, this time round, we only compute the dominant frequency once. This curve is then used to find *gaps* in the spectrogram, just as we do in Algorithm 12. Next we binarize the spectrogram, with a method similar to Algorithm 13. Finally, we extract contours using a Marching Squares[174] algorithm. After some filtering (predominantly based on contour length and x-range), we find closed contours describing female kiwi calls. An example of this process can be seen in Figure 5.20, where we plot closed contours on top of a binarized spectrogram. Though we have designed a process to extract such calls, we have yet to analyse the calls. This analysis will follow a similar concept to the male kiwis, except that we will try different algorithms to compute pairwise distances, such as Geometric Currents and SRVF Path-Straightening, that both focus on closed curves.

Validating Methods: Up until now, we have obtained successful classification results. However, as our sample size has been rather modest, our next step is to evaluate our methods on a larger dataset of male kiwi calls. Additionally, we will examine common methods that are used in classifying birdsong today, that do not utilize any shape analysis; and we will compare the results to ours. Henceforth, if the great success of our algorithms is repeated and validated, we can incorporate it onto the AviaNZ software[175].

[174] https://scikit-image.org/docs/0.8.0/api/skimage.measure.find_contours.html
[175] https://www.avianz.net/index.php

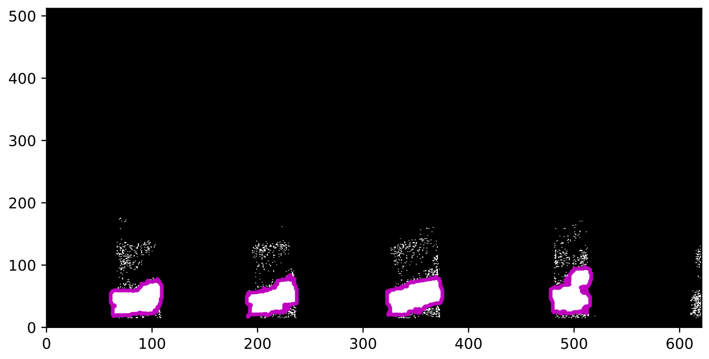


Figure 5.20: Female kiwi calls on a binarized spectrogram. Here, contours surrounding the calls are extracted using methods similar to those described earlier, concerning the male kiwi calls. The closed call-contours are plotted in magenta.

Significance

To the best of our knowledge, elastic shape analysis had never been applied to sound data until this very project; nor had it been applied to conservation prior to both this project and the kākāpō project (see Section 5.1). This is all the more reason to feel excited about the promising results (e.g., in Table 5.3) we have achieved so far. Though this is a *pilot* study, we have shown that we can successfully classify individual birds within a species by analysing the shapes of their calls, when plotted on spectrograms. Thus we truly believe that the tools we have made can one day play a significant role in acoustic species identification, used for the conservation of not only kiwis, but many other species too.

Chapter 6

Conclusion

In this thesis, we have explored the practical applications of elastic shape analysis on curves in $\mathbb{R}^2$, through a data science lens, with numerous projects on varying types of data. One common thing we have learnt across these various projects is that *shape* can certainly be used to analyse a dataset of objects.

Our research has resulted in novel applications of elastic shape analysis, innovative algorithms that can be used to process image and certain sound data, and overall, a new framework that can be used to analyse and classify object from their shapes, with the majority of the code freely available on open-source platforms[176]. In this final section, we summarize the work we have done in this thesis whilst highlighting the novel research, and finally, we outline the future of our research.

6.1 Summary

Applications of Shape Analysis

The datasets in our projects covered several questions and a wide range of areas, from growth curves to biological image datasets, and archaeological databases to audio data.

Beginning with our work on Greek vases, we learnt that methods of elastic shape analysis *outperform* the traditional geometric morphometrics method that is widely used by

[176]Our public GitHub repository, https://github.com/LittleAri/shapeClassification, provides code to extract contours from images, use elastic shape analysis on the contour data, such as distance computations, Karcher means, and code to perform classifications.

applied scientists, as well as experts in the field, when it comes to classification-related questions. Here, we also learnt that the analysis of shapes is best done within the SRVF framework, whether it is to compute distances that will later be incorporated into a machine learning classifier, or to cluster shapes or compute means (via a Karcher mean). Further work in this chapter also showed that Geometric Currents worked rather well in the classification of objects, particularly when shape differences were subtle.

In our last chapter, our two projects focused on a novel application of functional and shape data analysis, namely, in wildlife conservation. Our work, in collaboration with Kākāpō Recovery[177], resulted in a new tool that incorporated shape analysis to predict the health of Kākāpō chicks via the use of Karcher means. In our final project, we introduced the first real-world application of elastic shape analysis on raw sound data – more specifically, on recordings of kiwi calls. By combining shape analysis with machine learning, we designed an algorithm to classify individuals within a species by studying the shapes of their recorded calls.

Image Processing Endeavours

Throughout this entire thesis, we have made sure to state the importance of the image processing steps when it comes to shape analysis based on image datasets[178]. Whilst traditionally in literature there is little emphasis placed on such image processing techniques, in this thesis we have developed and optimised image processing methods specifically for each project's dataset. This led to a series of robust algorithms that we employed for our projects, and that can now be utilized by others, in further applications.

The most notable contribution to image processing in this thesis has been the automated binarization algorithm (as seen in Algorithm 5) aimed at segmenting an object centred in an image, whilst binarizing it. Though methods do exist that can obtain similar results, particularly with the use of neural networks, our method provides a much simpler solution that does not rely on the training of large datasets. This algorithm, or parts of it, were subsequently used in the data processing step for multiple projects throughout the thesis.

[177] `https://www.doc.govt.nz/our-work/kakapo-recovery/`

[178] Or even non-image datasets, such as the Kiwi call data, which eventually can be treated as images via the use of spectrograms, as seen in Section 5.2.1.

Another important image processing contribution appeared in our final project, which focused on spectrograms. Here, we designed a series of algorithms that incorporated classical tools from image processing such as contour extraction and image binarization with signal processing tools, in order to extract smooth harmonic curves from audio data.

Additional endeavours into the field of image processing and computer vision included our work on automatic vase-handle removal, and our technique of re-purposing an algorithm known for contour extraction (i.e. *Snakes*) for smoothing contours, by using a low number of iterations on an initial snake based on *un-smooth* contour.

6.2 Future Work

Throughout this thesis, we have discussed the future possibilities of our research in the relevant sections of our projects. Here, we highlight five future opportunities in particular:

Pairwise-Registration in the Product Space – An open problem in elastic shape analysis is on the pairwise registration of open curves in a product space. As discussed in Section 5.2.6, the implications of the defined mathematics can be used to register pairs of groups of curves. This will potentially lead to numerous applications, from analysing harmonics in audio data to pattern matching in zoological datasets.

Machine Learning & Elastic Shape Analysis – There remain many potential opportunities to explore regarding a combination of elastic shape analysis and machine learning. Whilst in this thesis we combined the two in order to classify objects based on geodesic distances, there are further possibilities. For example, we could look at incorporating an elastic metric within a different machine learning algorithm, such as k-means, using a Karcher mean instead of the standard Euclidean mean. Alternatively, we can explore more ways of connecting elastic metrics with deep learning, such as the work done in (Hartman et al., 2021), which trained a neural network model on elastic distances within the SRVF framework.

Large-scale Ancient Greek Vase Project – We mention this specific project in our future work as it has already been planned as a very large collaborative project. In this thesis, we saw how remarkably well we were able to classify vases, based solely on their outline shapes, by incorporating elastic shape analysis methods. Now, we want to evolve that project by working on a colossal dataset of Greek vases. This project will also require some work on the handle removal algorithms we previously designed, in order to avoid any manual manipulations to the images in this vast dataset.

Applications to 3D Surfaces – Throughout the entirety of this thesis, we have focused on two dimensional curves. Thus, a natural next step would be to consider 3D surfaces and curves in $\mathbb{R}^3$. Indeed, with the increased availability to 3D laser scanners, 3D data is becoming more plentiful. Moreover, there exist methods of elastic shape analysis that can certainly be applied to three dimensional data. This is the motivation for future work on the analysis of diverse applications of elastic shape analysis on 3D data, to answer various questions, such as classification.

Robust Contour Extraction Algorithms – Across our projects, we have worked on various image processing algorithms. Although these methods have worked well in our specific projects, we are mindful that they have not been examined much, in isolation, outside of our projects, apart from some *simple* tests, such as the experiment seen in Chapter 3. Thus our next step is to provide a concise comparative analysis with a larger dataset of images of various objects, on the methods that we have created, and on other popular methods in image processing, including this time, a neural network approach. By conducting a detailed experiment, we can utilize the results to improve the robustness of our algorithms. This work is significant as contour extraction tools are highly important in shape analysis. Henceforth, we will compile the code into a package or software, so that it can form part of our *guidebook* on the applications of shape analysis.

Guidebook for Applied Scientists – At the start of this thesis, we discussed our motivation for bringing shape analysis to a wider audience, outside of the shape-analysis community. Furthermore, we discussed the importance of a *guidebook* detailing how to analyse shape data with shape analysis, aimed at applied scientists. Such a guidebook would not only include details surrounding methods of elastic shape analysis, but it would also include the vital image processing steps, to process the shape data, as well as the machine learning steps, to utilize the results obtained from shape analysis. Whilst we hope the contents of this thesis can provide just that, in the future, we will compile a more concise handbook and make our code publicly available within a Python package, in order to *truly* showcase the significance of the applications of elastic shape analysis.

Bibliography

Abboud, M., Benzinou, A., Nasreddine, K., and Jazar, M. (2015). Robust statistical shape analysis based on the tangent shape space. *2015 IEEE International Conference on Image Processing (ICIP).*

Abdullah, N., Ngah, U. K., and Aziz, S. A. (2011). Image classification of brain MRI using support vector machine. In *2011 IEEE International Conference on Imaging Systems and Techniques*, pages 242–247. IEEE.

Adams, D. C., Rohlf, F. J., and Slice, D. E. (2004). Geometric morphometrics: ten years of progress following the 'revolution'. *Italian Journal of Zoology*, 71(1):5–16.

Amrane, M., Oukid, S., Gagaoua, I., and Ensari, T. (2018). Breast cancer classification using machine learning. In *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, pages 1–4. IEEE.

Anson, D. (2017). Analysis and classification of variations in the shape of Mycenaean pictorial amphoroid kraters from Cyprus and the Levant. *Mediterranean Archaeology*, 30:1–18.

Bauer, M., Bruveris, M., Marsland, S., and Michor, P. W. (2014). Constructing reparameterization invariant metrics on spaces of plane curves. *Differential Geometry and its Applications*, 34:139–165.

Bauer, M., Eslitzbichler, M., and Grasmair, M. (2015). Landmark-guided elastic shape analysis of human character motions. *arXiv preprint arXiv:1502.07666.*

Beazley, J. D. (1956). *Attic black-figure vase-painters.* Clarendon Press.

Bechshøft, T. Ø., Sonne, C., Rigét, F. F., Wiig, Ø., and Dietz, R. (2008). Differences in growth, size and sexual dimorphism in skulls of East Greenland and Svalbard polar bears (ursus maritimus). *Polar Biology*, 31(8):945–958.

Beg, M. F., Miller, M. I., Trouvé, A., and Younes, L. (2005). Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International Journal of Computer Vision*, 61(2):139–157.

Benn, J., Marsland, S., McLachlan, R. I., Modin, K., and Verdier, O. (2019). Currents and finite elements as tools for shape space. *Journal of Mathematical Imaging and Vision*, 61(8):1197–1220.

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:.

Bharath, K., Kurtek, S., Rao, A., and Baladandayuthapani, V. (2018). Radiologic image-based statistical shape analysis of brain tumours. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 67(5):1357–1378.

Bishop, C. M. (2006). Pattern recognition and machine learning. *Machine Learning*, 128(9).

Bloesch, H. (1940). *Formen attischer Schalen von Exekias bis zum Ende des strengen Stils.* Bümpliz, Benteli.

Bock, A. and Cotter, C. (2020). Space-time metamorphosis. *arXiv preprint arXiv:2005.08743.*

Bookstein, F., Chernoff, B., Elder, R., Humphries, J., Smith, G., and Strauss, R. (1985). The geometry of size and shape change, with examples from fishes. *Philadelphia: The Academy of Natural Sciences of Philadelphia.*

Bookstein, F. L. (1997). *Morphometric tools for landmark data.*

Bookstein, F. L. and Sampson, P. D. (1990). Statistical models for geometric components of shape change. *Communications in Statistics-Theory and Methods*, 19(5):1939–1972.

Brice, C. R. and Fennema, C. L. (1970). Scene analysis using regions. *Artificial Intelligence*, 1(3-4):205–226.

Bruce, B. and Marilyn, S. (1993). Music in theory and practice.

Brunner, N., Kühleitner, M., and Renner-Martin, K. (2021). Bertalanffy-Pütter models for avian growth. *PLOS ONE*, 16(4):e0250515.

Bruveris, M. (2016). Optimal reparametrizations in the square root velocity framework. *SIAM Journal on Mathematical Analysis*, 48(6):4335–4354.

Burns, B., Innes, J., and Day, T. (2012). The use and potential of pest-proof fencing for ecosystem restoration and fauna conservation in New Zealand. *Fencing for Conservation*, pages 65–90.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698.

Caselles, V., Kimmel, R., and Sapiro, G. (1995). Geodesic active contours. In *Proceedings of IEEE international conference on computer vision*, pages 694–699. IEEE.

Caselles, V., Kimmel, R., and Sapiro, G. (1997). Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79.

Chalana, V., Linker, D. T., Haynor, D. R., and Kim, Y. (1996). A multiple active contour model for cardiac boundary detection on echocardiographic sequences. *IEEE Transactions on Medical Imaging*, 15(3):290–298.

Cheng, J., Xie, B., Lin, C., and Ji, L. (2012). A comparative study in birds: call-type-independent species and individual recognition using four machine-learning methods and two acoustic features. *Bioacoustics*, 21(2):157–171.

Cho, M. H., Asiaee, A., and Kurtek, S. (2019). Elastic statistical shape analysis of biological structures with case studies: A tutorial. *Bulletin of Mathematical Biology*, 81(7):2052–2073.

Chopina, J., Miklavcic, S., and Lagaa, H. (2013). Selection of parameters in active contours for the phenotypic analysis of plants. *Proc. 20th Int. Congr. Modelling Simulation*, pages 510–516.

Christensen, G. E., Rabbitt, R. D., and Miller, M. I. (1996). Deformable templates using large deformation kinematics. *IEEE Transactions on Image Processing*, 5(10):1435–1447.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301.

Corfield, J., Gillman, L., and Parsons, S. (2008). Vocalizations of the North Island brown kiwi (apteryx mantelli). *The Auk*, 125(2):326–335.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cottam, Y., Merton, D. V., and Hendricks, W. (2006). Nutrient composition of the diet of parent-raised kakapo nestlings. *Notornis*, 53(1):90.

De Boor, C. (1978). *A practical guide to splines*, volume 27. springer-verlag New York.

De Cheveigné, A. and Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930.

De Rham, G. (1973). *Variétés différentiables: formes, courants, formes harmoniques*, volume 3. Editions Hermann.

Dhondt, A. and Lambrechts, M. M. (1992). Individual voice recognition in birds. *Trends in Ecology & Evolution*, 7(6):178–179.

Digby, A. (2013). *Whistling in the dark: an acoustic study of Little Spotted Kiwi.* PhD thesis, Victoria University of Wellington.

Digby, A., Bell, B. D., and Teal, P. D. (2013). Vocal cooperation between the sexes in little spotted kiwi (apteryx owenii). *Ibis*, 155(2):229–245.

Digby, A., Bell, B. D., and Teal, P. D. (2014). Vocal individuality of little spotted kiwi (apteryx owenii). *Emu-Austral Ornithology*, 114(4):326–336.

Dinov, I. D., Petrosyan, P., Liu, Z., Eggert, P., Zamanyan, A., Torri, F., Macciardi, F., Hobel, S., Moon, S. W., Sung, Y. H., et al. (2014). The perfect neuroimaging-genetics-computation storm: collision of petabytes of data, millions of hardware devices and thousands of software tools. *Brain Imaging and Behavior*, 8(2):311–322.

Dryden, I. L. and Mardia, K. V. (2016). *Statistical shape analysis: with applications in R*, volume 995. John Wiley & Sons.

Dupuis, P., Grenander, U., and Miller, M. I. (1998). Variational problems on flows of diffeomorphisms for image matching. *Quarterly of Applied Mathematics*, 56(3):587–600.

Eason, D. K. and Moorhouse, R. J. (2006). Hand-rearing kakapo (strigops habroptilus), 1997-2005. *Notornis*, 53(1):116.

Elliott, G. P., Merton, D. V., and Jansen, P. W. (2001). Intensive management of a critically endangered species: the kakapo. *Biological Conservation*, 99(1):121–133.

Engelbrecht, D., De Waal, D., Du Plooy, D., Theron, N., Turner, A., and Wilkinson, S. (2007). Growth curve analysis of hand-reared southern and northern ground hornbill nestlings.

Eslitzbichler, M. (2015). Modelling character motions on infinite-dimensional manifolds. *The Visual Computer*, 31(9):1179–1190.

Falls, J. B. (1982). Individual recognition by sounds in birds. *Acoustic Communication in Birds*, 2:237–278.

Fidler, A. E., Lawrence, S. B., and McNatty, K. P. (2008). An hypothesis to explain the linkage between kakapo (strigops habroptilus) breeding and the mast fruiting of their food trees. *Wildlife Research*, 35(1):1–7.

Firmansyah, Z., Herdiyeni, Y., Silalahi, B. P., and Douady, S. (2016). Landmark analysis of leaf shape using polygonal approximation. In *IOP Conference Series: Earth and Environmental Science*, volume 31, page 012018. IOP Publishing.

Fix, E. and Hodges Jr, J. L. (1952). Discriminatory analysis-nonparametric discrimination: Small sample performance. Technical report, California Univ Berkeley.

Fretwell, P. T., Scofield, P., and Phillips, R. A. (2017). Using super-high resolution satellite imagery to census threatened albatrosses. *Ibis*, 159(3):481–490.

Gerhard, D. et al. (2003). *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, SK, Canada.

Getoor, L. and Taskar, B. (2019). *Introduction to Statistical Relational Learning*. MIT Press.

Glaunes, J., Qiu, A., Miller, M. I., and Younes, L. (2008). Large deformation diffeomorphic metric curve mapping. *International Journal of Computer Vision*, 80(3):317–336.

Gong, S. and Newman, T. S. (2013). A corner feature sensitive marching squares. In *2013 Proceedings of IEEE Southeastcon*, pages 1–6. IEEE.

Goodall, C. (1991). Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(2):285–321.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gower, J. (1971). Statistical methods of comparing different multivariate analyses of the same data. *Mathematics in the Archaeological and Historical Sciences*, 138:149.

Grenander, U. and Miller, M. I. (1998). Computational anatomy: An emerging discipline. *Quarterly of Applied Mathematics*, 56(4):617–694.

Grove, K. and Karcher, H. (1973). How to conjugate C^1-close group actions. *Mathematische Zeitschrift*, 132(1):11–20.

Gündemir, M. G., Szara, T., Spataru, C., Demircioglu, I., Turek, B., Petrovas, G., and Spataru, M. C. (2022). Shape differences of the carina sterni in birds of various locomotion types. *Anatomia, Histologia, Embryologia*.

Gunz, P. and Mitteroecker, P. (2013). Semilandmarks: a method for quantifying curves and surfaces. *Hystrix, the Italian Journal of Mammalogy*, 24(1):103–109.

Hansen, C. D. and Johnson, C. R. (2011). *Visualization handbook*. Elsevier.

Harper, G. A., Elliott, G. P., Eason, D. K., and Moorhouse, R. J. (2006). What triggers nesting of kakapo (strigops habroptilus)? *Notornis*, 53(1):160.

Hartman, E., Sukurdeep, Y., Charon, N., Klassen, E., and Bauer, M. (2021). Supervised deep learning of elastic SRV distances on the shape space of curves. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4425–4433.

Höllig, K. and Hörner, J. (2013). *Approximation and modeling with B-splines*. SIAM.

Holm, D. D., Marsden, J. E., and Ratiu, T. S. (1998). The Euler–Poincaré equations and semidirect products with applications to continuum theories. *Advances in Mathematics*, 137(1):1–81.

Holm, D. D., Marsden, J. E., and Ratiu, T. S. (1999). The Euler-Poincaré equations in geophysical fluid dynamics. *arXiv preprint chao-dyn/9903035*.

Huang, S., Jia, J., Cao, R., Li, G., Cheng, M., and Wu, Y. (2011). Automatic segmentation of the body and the spinal canal in CT images based on a priori information. In *2011 5th International Conference on Bioinformatics and Biomedical Engineering*, pages 1–4. IEEE.

Immerwahr, H. R. (1984). The signatures of Pamphaios. *American Journal of Archaeology*, pages 341–352.

Ivins, J. and Porrill, J. (1995). Everything you always wanted to know about snakes (but were afraid to ask). *Artificial Intelligence*, 2000.

Iwata, H. and Ukai, Y. (2002). Shape: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. *Journal of Heredity*, 93(5):384–385.

Joháczi, S. (2018). A new method in attribution? attempts of the employment of geometric morphometrics in the attribution of Late Archaic Attic lekythoi. *Dissertationes Archaeologicae*, pages 371–418.

Joshi, S. H., Klassen, E., Srivastava, A., and Jermyn, I. (2007). A novel representation for Riemannian analysis of elastic curves in rn. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE.

Joshi, S. H., Prieto-Marquez, A., and Parker, W. C. (2011). A landmark-free method for quantifying biological shape variation. *Biological Journal of the Linnean Society*, 104(1):217–233.

Julina, J. K. J. and Sharmila, T. S. (2017). Facial recognition using histogram of gradients and support vector machines. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–5. IEEE.

Kahl, S., Wood, C. M., Eibl, M., and Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, 61:101236.

Kambhatla, N. and Leen, T. K. (1997). Dimension reduction by local principal component analysis. *Neural computation*, 9(7):1493–1516.

Karcher, H. (1977). Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541.

Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of computer vision*, 1(4):321–331.

Kendall, D. G. (1984). Shape manifolds, Procrustean metrics, and complex projective spaces. *Bulletin of the London Mathematical Society*, 16(2):81–121.

Kendall, D. G., Barden, D., Carne, T. K., and Le, H. (2009). *Shape and shape theory*, volume 500. John Wiley & Sons.

Klassen, E. and Srivastava, A. (2006). Geodesics between 3D closed curves using path-straightening. In *European conference on computer vision*, pages 95–106. Springer.

Klingenberg, C. P. (2020). Walking on Kendall's shape space: Understanding shape spaces and their coordinate systems. *Evolutionary Biology*, 47(4):334–352.

Kriegl, A. and Michor, P. W. (1997). *The convenient setting of global analysis*, volume 53. American Mathematical Soc.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Kuhl, F. P. and Giardina, C. R. (1982). Elliptic fourier features of a closed contour. *Computer graphics and image processing*, 18(3):236–258.

Kühleitner, M., Brunner, N., Nowak, W.-G., Renner-Martin, K., and Scheicher, K. (2019). Best-fitting growth curves of the von Bertalanffy-Pütter type. *Poultry Science*, 98(9):3587–3592.

Kuo, T.-Y., Lai, Y.-Y., and Lo, Y.-C. (2010). A novel image binarization method using hybrid thresholding. In *2010 IEEE International Conference on Multimedia and Expo*, pages 608–612. IEEE.

Kurtek, S. and Srivastava, A. (2014). Handwritten text segmentation using elastic shape analysis. In *2014 22nd International Conference on Pattern Recognition*, pages 2501–2506. IEEE.

Laborde, J., Srivastava, A., and Zhang, J. (2011). Structure-based RNA function prediction using elastic shape analysis. *2011 IEEE International Conference on Bioinformatics and Biomedicine.*

Laga, H., Kurtek, S., Srivastava, A., and Miklavcic, S. J. (2014). Landmark-free statistical analysis of the shape of plant leaves. *Journal of Theoretical Biology*, 363:41–52.

Lahiri, S., Robinson, D., and Klassen, E. (2015). Precise matching of PL curves in $\mathbb{R}^N$ in the square root velocity framework. *arXiv preprint arXiv:1501.00577.*

Lakshminarayanan, B., Raich, R., and Fern, X. (2009). A syllable-level probabilistic framework for bird species identification. In *2009 International Conference on Machine Learning and Applications*, pages 53–59. IEEE.

Le, H. and Kume, A. (2000). The Fréchet mean shape and the shape of the means. *Advances in Applied Probability*, 32(1):101–113.

Lee, J. (2010). *Introduction to topological manifolds*, volume 202. Springer Science & Business Media.

Leighton, L. R. (2011). Analyzing predation from the dawn of the Phanerozoic. *Quantifying the Evolution of Early Life*, pages 73–109.

Leoni, G. (2017). *A first course in Sobolev spaces.* American Mathematical Soc.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169.

Mackay, A., Joháczi, S., Salili-James, A., Leroi, A. M., Mannack, T., and Marsland, S. (2021). Measuring the shapes of ancient Greek vases. *Available at SSRN 4012965.*

Mackay, E. A. (2010). *Tradition and originality: a study of Exekias.* Archaeopress.

Magbayao, R. A., Arboleda, E. R., and Galas, E. M. (2020). Identification of Asian green mussel perna viridis' sex using image processing, fuzzy logic and k–nearest neighbor. *Int. J. Sci. Technol. Res*, 9(01).

Maple, C. (2003). Geometric design and space planning using the marching squares and marching cube algorithms. In *2003 international conference on geometric modeling and graphics, 2003. Proceedings*, pages 90–95. IEEE.

Mardia, K. and Dryden, I. (1989). The statistical analysis of shape data. *Biometrika*, 76(2):271–281.

Marsland, S. (2015). *Machine learning: an algorithmic perspective.* CRC press.

Marsland, S., Priyadarshani, N., Juodakis, J., and Castro, I. (2019). AviaNZ: A future-proofed program for annotation and recognition of animal sounds in long-time field recordings. *Methods in Ecology and Evolution*, 10(8):1189–1195.

Marsland, S. and Shardlow, T. (2017). Langevin equations for landmark image registration with uncertainty. *SIAM Journal on Imaging Sciences*, 10(2):782–807.

Marsland, S. and Sommer, S. (2020). Riemannian geometry on shapes and diffeomorphisms: Statistics via actions of the diffeomorphism group. In *Riemannian Geometric Statistics in Medical Image Analysis*, pages 135–167. Elsevier.

Marsland, S. and Twining, C. J. (2004). Constructing diffeomorphic representations for the groupwise analysis of nonrigid registrations of medical images. *IEEE Transactions on Medical Imaging*, 23(8):1006–1020.

Matuk, J., Mohammed, S., Kurtek, S., and Bharath, K. (2020). Biomedical applications of geometric functional data analysis. In *Handbook of Variational Methods for Nonlinear Geometric Data*, pages 675–701. Springer.

Michor, P. W. and Mumford, D. (2007). An overview of the Riemannian metrics on spaces of curves using the Hamiltonian approach. *Applied and Computational Harmonic Analysis*, 23(1):74–113.

Miller, M. I. (2004). Computational anatomy: shape, growth, and atrophy comparison via diffeomorphisms. *NeuroImage*, 23:S19–S33.

Minaee, S., Boykov, Y. Y., Porikli, F., Plaza, A. J., Kehtarnavaz, N., and Terzopoulos, D. (2021). Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Minchin, D., Maguire, C., and Rosell, R. (2003). The zebra mussel (dreissena polymorpha pallas) invades ireland: human mediated vectors and the potential for rapid intranational dispersal. In *Biology and Environment: Proceedings of the Royal Irish Academy*, pages 23–30. JSTOR.

Mio, W., Srivastava, A., and Joshi, S. (2007). On shape of plane elastic curves. *International Journal of Computer Vision*, 73(3):307–324.

Miskelly, C. M. and Powlesland, R. G. (2013). Conservation translocations of New Zealand birds, 1863–2012. *Notornis*, 60(1):3–28.

Mitteroecker, P. and Gunz, P. (2009). Advances in geometric morphometrics. *Evolutionary Biology*, 36(2):235–247.

Moffett, J. (1992). The beazley archive: Making a humanities database accessible to the world. *Bulletin of the John Rylands Library*, 74(3):39–52.

Mosier, C. I. (1939). Determining a simple structure when loadings for certain tests are known. *Psychometrika*, 4(2):149–162.

Mostofi, F. and Khashman, A. (2014). Intelligent recognition of ancient Persian cuneiform characters. In *IJCCI (NCTA)*, pages 119–123.

Mottini, A., Descombes, X., and Besse, F. (2014). Axonal tree classification using an elastic shape analysis based distance. In *2014 IEEE 11th International symposium on biomedical imaging (ISBI)*, pages 850–853. IEEE.

Mouine, S., Yahiaoui, I., and Verroust-Blondet, A. (2013). A shape-based approach for leaf classification using multiscaletriangular representation. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 127–134.

Munshi, P. and Mitra, S. K. (2012). A rough-set based binarization technique for fingerprint images. In *2012 IEEE International Conference on Signal Processing, Computing and Control*, pages 1–6. IEEE.

Niblack, W. (1985). *An introduction to digital image processing.* Strandberg Publishing Company.

Norman, M., Jonathan, K., and Kate, E. (2013). Geometric morphometric approaches to acoustic signal analysis in mammalian biology. *Virtual Morphology and Evolutionary Morphometrics in the New Millenium.*, page 110.

Okumura, M. and Araujo, A. G. (2019). Archaeology, biology, and borrowing: A critical examination of geometric morphometrics in archaeology. *Journal of Archaeological Science*, 101:149–158.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66.

Ovaskainen, O., Moliterno de Camargo, U., and Somervuo, P. (2018). Animal sound identifier (asi): software for automated identification of vocal animals. *Ecology letters*, 21(8):1244–1254.

Pal, M., Paul, D., and Saha, G. (2018). Synthetic speech detection using fundamental frequency variation and spectral features. *Computer Speech & Language*, 48:31–50.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Pennec, X. (1999). Probabilities and statistics on Riemannian manifolds: Basic tools for geometric measurements. In *NSIP*, volume 3, pages 194–198. Citeseer.

Peters, M. A., Hamilton, D., Eames, C., Innes, J., and Mason, N. W. (2016). The current state of community-based environmental monitoring in New Zealand. *New Zealand Journal of Ecology*, 40(3):279–288.

Prabhu, K. M. (2014). *Window functions and their applications in signal processing.* Taylor & Francis.

Priddy, K. L. and Keller, P. E. (2005). *Artificial neural networks: an introduction*, volume 68. SPIE press.

Prieto-Márquez, A. and Joshi, S. H. (2015). Morphological variation of pelvic skeletal elements of hadrosaurid dinosaurs quantified using Riemannian analysis of elastic curves. In *Biological Shape Analysis : Proceedings of the 3rd International Symposium*, pages 138–155. World Scientific.

Priyadarshani, N., Castro, I., and Marsland, S. (2018). The impact of environmental factors in birdsong acquisition using automated recorders. *Ecology and Evolution*, 8(10):5016–5033.

Priyadarshani, N., Marsland, S., Castro, I., and Punchihewa, A. (2016). Birdsong denoising using wavelets. *PLOS ONE*, 11(1):e0146790.

Priyadarshani, N., Marsland, S., Juodakis, J., Castro, I., and Listanti, V. (2020). Wavelet filters for automated recognition of birdsong in long-time field recordings. *Methods in Ecology and Evolution*, 11(3):403–417.

Qian, K., Zhang, Z., Ringeval, F., and Schuller, B. (2015). Bird sounds classification by large scale acoustic features and extreme learning machine. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1317–1321. IEEE.

Ralph, C. J., Droege, S., and Sauer, J. R. (1995). Managing and monitoring birds using point counts: standards and applications. *In: Ralph, C. John; Sauer, John R.; Droege, Sam, technical editors. 1995. Monitoring bird populations by point counts. Gen. Tech. Rep. PSW-GTR-149. Albany, CA: US Department of Agriculture, Forest Service, Pacific Southwest Research Station: p. 161-168*, 149.

Rath, T. M. and Manmatha, R. (2003). Word image matching using dynamic time warping. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II. IEEE.

Robinson, D. T. (2012). *Functional data analysis and partial shape matching in the square root velocity framework*. PhD thesis, Florida State University.

Rocha, P. C. and Romano, P. S. (2021). The shape of sound: A new R package that crosses the bridge between bioacoustics and geometric morphometrics. *Methods in Ecology and Evolution*, 12(6):1115–1121.

Rohlf, F. J. and Marcus, L. F. (1993). A revolution in morphometrics. *Trends in Ecology & Evolution*, 8(4):129–132.

Rousselle, J.-J., Vincent, N., and Verbeke, N. (2003). Genetic algorithm to set active contour. In *International Conference on Computer Analysis of Images and Patterns*, pages 345–352. Springer.

Rydning, D. R.-J. G.-J. (2018). The digitization of the world from edge to core. *Framingham: International Data Corporation*, page 16.

Salili-James, A., Mackay, A., Rodriguez-Alvarez, E., Rodriguez-Perez, D., Mannack, T., Rawlings, T. A., Palmer, A. R., Todd, J., Riutta, T. E., Macinnis-Ng, C., et al. (2021). Classifying organisms and artefacts by their shapes. *arXiv preprint arXiv:2109.00920*.

Schneider, S., Taylor, G. W., and Kremer, S. (2018). Deep learning object detection methods for ecological camera trap data. In *2018 15th Conference on computer and robot vision (CRV)*, pages 321–328. IEEE.

Selden Jr, R. Z. (2019). Ceramic morphological organisation in the Southern Caddo Area: the Clarence H. Webb collections. *Journal of Cultural Heritage*, 35:41–55.

Sheets, H. D., Covino, K. M., Panasiewicz, J. M., and Morris, S. R. (2006). Comparison of geometric morphometric outline methods in the discrimination of age-related differences in feather shape. *Frontiers in Zoology*, 3(1):1–12.

Sibson, R. (1978). Studies in the robustness of multidimensional scaling: Procrustes statistics. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(2):234–238.

Smola, A. and Vishwanathan, S. (2008). Introduction to machine learning. *Cambridge University, UK*, 32(34):2008.

Snell, O. (1892). Die abhängigkeit des hirngewichtes von dem körpergewicht und den geistigen fähigkeiten. *Archiv für Psychiatrie und Nervenkrankheiten*, 23(2):436–446.

Srivastava, A., Klassen, E., Joshi, S. H., and Jermyn, I. H. (2011a). Shape analysis of elastic curves in Euclidean spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(7):1415–1428.

Srivastava, A. and Klassen, E. P. (2016). *Functional and shape data analysis*, volume 1. Springer.

Srivastava, A., Wu, W., Kurtek, S., Klassen, E., and Marron, J. S. (2011b). Registration of functional data using Fisher-Rao metric. *arXiv preprint arXiv:1103.3817*.

Strait, J., Kurtek, S., Bartha, E., and MacEachern, S. N. (2017). Landmark-constrained elastic shape analysis of planar curves. *Journal of the American Statistical Association*, 112(518):521–533.

Su, Z., Klassen, E., and Bauer, M. (2017). The square root velocity framework for curves in a homogeneous space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–10.

Suzuki, S. et al. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46.

Telgarsky, R. (2013). Dominant frequency extraction. *arXiv preprint arXiv:1306.0103*.

Thompson, D. W. (1917). *On growth and form*. Cambridge University Press.

Trouvé, A. (1995). An infinite dimensional group approach for physics based models in pattern recognition. *International Journal of Computer Vision - IJCV*.

Trouvé, A. and Younes, L. (2005). Metamorphoses through Lie group action. *Foundations of Computational Mathematics*, 5(2):173–198.

Tucker, J. D., Wu, W., and Srivastava, A. (2013). Analysis of signals under compositional noise with applications to SONAR data. *IEEE Journal of Oceanic Engineering*, 39(2):318–330.

Twining, C. J. and Marsland, S. (2003). Constructing diffeomorphic representations of non-rigid registrations of medical images. In *Biennial International Conference on Information Processing in Medical Imaging*, pages 413–425. Springer.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and the scikit-image contributors (2014). scikit-image: image processing in Python. *PeerJ*, 2:e453.

Venier, L., Mazerolle, M., Rodgers, A., McIlwrick, K., Holmes, S., and Thompson, D. (2017). Comparison of semiautomated bird song recognition with manual detection of recorded bird song samples. *Avian Conservation and Ecology*, 12(2).

Wang, J., Qian, W., and Chen, G. (2021). Combining quantitative analysis with an elliptic Fourier descriptor: A study of pottery from the Gansu-Zhanqi site based on 3D scanning and computer technology. *Journal of Archaeological Science: Reports*, 36:102897.

Wyvill, G. and Anson, D. (2004). Extracting measurements from existing photographs of ancient pottery. In *Proceedings Computer Graphics International, 2004.*, pages 614–617. IEEE.

Yang, Q. and Marchant, J. A. (1996). Accurate blemish detection with active contour models. *Computers and Electronics in Agriculture*, 14(1):77–89.

Younes, L. (1998). Computable elastic distances between shapes. *SIAM Journal on Applied Mathematics*, 58(2):565–586.

Younes, L. (2010). *Shapes and diffeomorphisms*, volume 171. Springer.

Zadeh, L. A. (1996). Fuzzy sets. In *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, pages 394–432. World Scientific.

Zhang, M. and Fletcher, P. T. (2015). Finite-dimensional Lie algebras for fast diffeomorphic image registration. In *International conference on information processing in medical imaging*, pages 249–260. Springer.

Zhang, Z. (2016). Derivation of backpropagation in convolutional neural network (CNN). *University of Tennessee, Knoxville, TN.*

Appendix A

Kākāpō Classification Code

In light of our project in collaboration with Kākāpō Recovery[179], detailed in Section 5.1 of this thesis, we present a summary of the algorithms that make up our new Kākāpō Health tool.

In the following section, we detail the inputs and outputs of some of the functions used in the classification code. For more details regarding this code, and for examples, we refer the reader to our Github repository, kakapoClassification[180][181].

Note that in the following code, boolean type variables (`bool`), take in 1 and 0 for `True` / `False`.

[179] `https://www.doc.govt.nz/our-work/kakapo-recovery/`

[180] `https://github.com/LittleAri/kakapoClassification`

[181] As this a private repository between us and the Kākāpō Recovery team, please contact us for access.

A.1 Algorithms

A.1.1 Sampling Functions

The aim of the functions in this subsection is to create a sample of chicks and their weights. Such samples can be used to compute Karcher mean templates and so on.

Algorithm 18: createWeightsSample

```
createWeightsSample(birdNames, weightsData, metadata, sex="Female",
 sample=1, rimu=0, HR=0, minAge=0, maxAge=75, sampLength=40,
 leeway=0.75, minLength=3, HRInbetween=0, minMaxAge=0, healthFilter=0,
 healthFilterColumn="health")
```

Weights/ages from chicks in a sample, within a given range.

Parameters:

- `birdNames` - list of bird names in original sample.
- `weightsData (pd.DataFrame)` - all weights data.
- `metadata (pd.DataFrame)` - all meta data (e.g. **ripeRimu** information etc).
- `sex` - sex filter for sample birds. Set `sex=0` to not filter by sex.
- `sample (bool)` - Set `sample=1` to sample the ages (using 11). Else, the original weights and ages in the given age range, in the filtered sample will be returned.
- `rimu` - Rimu filter. Set `rimu='Y'` to filter for chicks born during rimu season etc.
- `HR` - Hand-rearing filter. Set `HR='Y'` to filter for hand-reared chicks etc.
- `minAge (np.float)` - minimum age of chick.
- `maxAge (np.float)` - maximum age of chick.
- `sampLength (np.int)` - number of samples in the age template used for sampling ages. This is only used if `sample==1`. Note that if the sampled age of a chick is less than `sampLength`, then it won't be included.
- `leeway (np.float)` - If `sample==1`, we filter for chicks whose first values in their sampled ages is within `minAge` ± `leeway`, and last is within `maxAge` ± `leeway`.
- `HRInbetween (bool)` - If `HRInbetween!=0`, then those with handReared status "YN", will be included in the `HR` filter. For example, if `HR==''Y'' and HRInbetween==1` then sample will be filtered for chicks with **handReared**='Y' or **handReared**='YN'.

- `minMaxAge (np.float)` - filters for chicks whose max age is greater than the minMaxAge. This is only used when `sample=0`.
- `healthFilter (bool)` - only includes chicks in sample whose general health is set to 1. To not have a health filter, set `healthFilter=0`.
- `healthFilterColumn (str)` - the name of the health column that `healthFilter` uses to filter the sample.

`return sampledWeights` - M weights and ages of chicks in new sample.
`return sampleNames` - names of chicks in new sample.
`return sampLength`

A.1.2 Defining Boundaries / Health

In this subsection, we detail out the functions used to create boundaries, as seen earlier. We also present the functions used to define a health classification, for example, if a certain number of consecutive points in a growth curve fall outside of a boundary.

Algorithm 19: createBounds

`createBounds(KarcherMean,standardDeviation,k=1.5,percentageScale=0)`

Create boundary around a mean.

Parameters:

- `KarcherMean (np.ndarray)` - Karcher mean, of size $(2, M)$. Note that this could be replaced by any desired mean.
- `standardDeviation (np.ndarray)` - one-dimensional array consisting of the standard deviation based on the aligned weights (of length M).
- `k (np.float)` - scalar used to create the boundary.
- `percentageScale (bool)` - if `percentageScale == 1`, the function will create a boundary based on a multiple of `KarcherMean` using `k`. Otherwise, the boundary is created using `KarcherMean` $\pm(k\times$`standardDeviation`$)$.

`return xBoundary (list)` - x values of the boundary, of size $(1, 2M)$.

`return yBoundary (list)` - y values of the boundary.

Algorithm 20: defineHealth

`defineHealth( T, F, KMlength, consecutiveCount, xBoundary, yBoundary, maxRatio=0.5, lowerBound=0, printPoints=0)`

Initial health prediction based on the boundary created in 19

Parameters:

- `T` - list of ages of a chick (asc.).
- `F` - list of weights corresponding to the ages in T.
- `KMlength (int)` - length of Karcher Mean, (i.e. half the size of the boundary, M).
- `consecutiveCount (int)` - number of consecutive points used to define health. If `consecutiveCount` consecutive points fall outside of the boundary, then this causes the initial classification to be 0 (unhealthy) etc.
- `xBoundary` - x values of boundary.
- `yBoundary` - y values of boundary.
- `maxRatio` - this parameter is only used if the number of values in `T` is less than `consecutiveCount`. In this case, if the ratio of points outside of the boundary and total points, is greater than `maxRatio`, the initially health classification would be 0.
- `lowerBound (bool)` - if this parameter is set to 1, then we would only check whether values lie underneath the boundary, instead of checking whether it falls outside of the boundary as a whole.
- `printPoints (bool)` - used only if you are interested in seeing the age of the chick at which it had the relevant number of consecutive points outside of the given boundary. Shows the first and last instance that this occurred. [Can only be seen for chicks classed as unhealthy, except those that used `maxRatio` for their classification.]

`return predictedHealth (bool)` - initial health classification (1 for *healthy* etc).

`return consecutivePoints (list)` - consecutive points, empty unless `printPoints=1`.

Algorithm 21: extraFilter

`extraFilter(mean, x, y, currentPrediction, majorityScalar=0.5, maxAge=75, minMeanValue=0, boundScalar=0.75)`

Updated health prediction, based on result of 20 and further conditions.

Parameters:

- `mean` - Mean, where `mean[0]` holds the x values and `mean[1]` the y values.
- `x (np.ndarray)` - ages of a chick (asc).
- `y` - list of weights of a chick at the ages listed in `x`.
- `currentPrediction (bool)` - initial health classification as defined by 20.
- `majorityScalar (np.float)` - used as an additional filter for chicks with initial health classification 0. If the ratio of the second half of the growth curve being over the mean is greater than `majorityScalar`, along with some other conditions, then the health classification is reverted to 1.
- `maxAge (np.float)` - If the initial classification is 0, but if `max(x)<maxAge/5`, then the initial classification is accepted no matter what.
- `minMeanValue (np.float)` - the minimum value that we accept a point in the mean to be. Can be used as a safety net for negative numbers in the mean for example.
- `boundScalar (np.float)` - If a chick has an initial classification of 1, but falls under a new lower bound (`mean - (mean`$\times$`boundScalar)`), then its health is reclassified as 0. Note that this step can only be used when `min(x)>=10` days.

`return newPredictedHealth (bool)` - updated health classification.

A.1.3 Computing Means

The following is used to compute mean weights from any desired group (e.g. Females, non-Rimu etc). It can be used to create the pre-computed Karcher means and optimal variables.

Algorithm 22: computeMeans

```
computeMeans(birdNames, birdWeights, metaData, ripe, hr, birdSex,
 minAge=0, maxAge=75, sampLength=40, leeway=0.75, stdLBound=0.005,
 kScalar=2,showBounds=0)
```

Compute mean, Karcher mean, aligned weights etc, of groups of chicks.

Parameters:

- `birdNames` - list of chick names.
- `birdWeights (pd.DataFrame)` - age and weights of chicks in sample.
- `metaData (pd.DataFrame)` - meta details of all chicks (e.g. **sex**, **ripeRimu** etc).
- `ripe` - **ripeRimu** filter, e.g. "Y" to filter for those born in Rimu season.
- `hr` - **handReared** filter, e.g. "N" to filter for non-hand-reared chicks.
- `birdSex` - **sex** filter.
- `minAge (np.float)` - minimum age of chick in growth curve.
- `maxAge (np.float)` - maximum age of chick in growth curve.
- `sampLength (np.int)` - length of template in sampling function.
- `leeway (np.float)` - refers to the `leeway` variable in (18).
- `stdLBound (np.float)` - minimum value in standard deviation (can be used as a safety net to exclude negative numbers or zeros).
- `kScalar (np.float)` - scalar used to create boundary with the Karcher mean and standard deviation (as seen in 19).
- `showBounds (bool)` - `if showBounds==0`, then the previous parameter is ignored and the boundary ranges won't be computed, and an empty list is returned instead.

`return testWeights` - ages and weights of chicks in new sample (training and testing).

`return testNames` - list of names of chicks in new sample.

`return mean` - mean weights in new sample. `mean[0]` for x values, and `mean[1]` for y.

`return KarcherMean` - Karcher mean of chicks in training sample. Size `(2,sampLength)`.

`return averageGamma` - average gamma function from training sample.

`return standardDeviation` - standard deviation of weights in new sample.

`return alignedWeights` - aligned weights of chicks in new sample.

`return boundary` - boundary curve, size `(2,2×sampLength)`.

A.1.4 Health Classification

These are the main functions in the health classification code. The former is used to train and test the prediction algorithm, whilst being used to test out an array of different parameter options. The latter function is used to predict the health of one chick, and would likely form the basis of any classification web-page.

Algorithm 23: trainTestKakapo

```
trainTestKakapo(birdNames, birdWeights, metaData, healthDetails,
 birdSex, ripe=0, hr=0, method=1.4, minAge=0, maxAge=75, sampLength=40,
 leeway=0.75, minLength=3, HRInbetween=0, excludeOutliers=1,
 excludeTraining=0, beta=0.15, average="weighted",
 percentageScalars=np.linspace(0.1, 0.5, 9), stdScalars=np.linspace(1.5,
 4.5, 13), distanceLimitScalars=np.linspace(0, 5, 11),
 consecutiveNumbers=np.linspace(3, 12, 10), percentageScale=0,
 lowerBound=0, testAlign=1, stdLowerBound=0.005, majorityScalar=0.5,
 printPoints=0, stdRound=0, minMaxAge=0, extraFilterScalar=0.75)
```

Function to train and test methods, and optimise parameters.

Parameters:

- `birdNames` - list of chick names.
- `birdWeights (pd.DataFrame)` - age and weights of chicks in sample.
- `metaData (pd.DataFrame)` - meta details of all chicks (e.g. **sex**, **ripeRimu** etc).
- `healthDetails (dict)` - health details e.g. `healthDetails[chick][‘healthy’]=1`.
- `birdSex` - **sex** filter.
- `ripe` - **ripeRimu** filter, e.g. “Y” to filter for those born in Rimu season.
- `hr` - **handReared** filter, e.g. “N” to filter for non-hand-reared chicks.
- `method` - Use `method=1.4` for our best method, or `method=0` for the original method.
- `minAge (np.float)` - minimum age of chick in growth curve.
- `maxAge (np.float)` - maximum age of chick in growth curve.

[182] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
[183] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

- `sampLength (np.int)` - length of template in sampling function (11).
- `leeway (np.float)` - refers to the `leeway` variable in (18).
- `minLength (np.int)` - minimum length of growth curve to predict health on.
- `HRInbetween (bool)` - refers to the `HRInbetween` variable in (18).
- `excludeOutliers (bool)` - excludes outliers in standard deviation computations.
- `excludeTraining (bool)` - excludes training sample when testing.
- `beta (np.float)` - refers to the `beta` variable in an F_β-score (see Equation (4.4)).
- `average` - refers to the `average` variable required in `sklearn`'s precision[182]and recall[183]scores, used in the computation of the F_β-score.
- `percentageScalars` - list containing percentage scalars to test for the boundary (e.g. used for `method=0`, in 19).
- `stdScalars` - list containing standard deviation scalars to test for the boundary (e.g. used for `method=1.4`, in 19).
- `distanceLimitScalars` - distance scalars. Used only when `method=2` or `method=3`.
- `consecutiveNumbers` - list of integers used for `consecutiveCount` variable in 20.
- `percentageScale (bool)` - option to use base boundary on percentage of mean, rather than the standard deviation.
- `lowerBound (bool)` - refers to `lowerBound` in 20.
- `testAlign (bool)` - if set to 0, testing will be done on the non-aligned weights.
- `stdLowerBound (np.float)` - minimum accepted value for the standard deviation.
- `majorityScalar (np.float)` - refers to the `majorityScalar` variable in 21.

- `printPoints (bool)` - refers to the `printPoints` variable in 20.
- `stdRound (bool)` - if set to 1, then the mean used 21 will be decreased by a quarter of the standard deviation.
- `minMaxAge (np.float)` - minimum maximum age of chicks in samples.
- `extraFilterScalar (np.float)` - refers to the `boundScalar` variable in 21.

`return classification` - true and predicted health classification.
`return scoresAndVariables (dict)` - classification test details e.g. top F_β.
`return outliers (np.int)` - total aligned growth curve outliers.
`return rainNames` - List of chicks in training sample.
`return testNames` - List of chicks in testing sample.

Algorithm 24: predictHealth

```
predictHealth(Name, birdNames, metaData, birdWeights, optimalVariables,
 method=1.4, T=0, F=0, HRInbetween="N", useHRBackUp=0, minAge=0,
 maxAge=75, sampLength=40, birdSex=0, ripe=0, hr=0, lowerBound=0,
 testAlign=1, percentageScale=0, percentageScalar=0,
 stdLowerBound=0.005, majorityScalar=0.5, stdRound=0,
 backupSex="Female", useBackUp=0, KarcherMeanDetails=0, optVariables=[],
 extraFilterScalar=0.75)
```

Predict health of chick.

Parameters:

- `Names` - name of specific chick to predict health.
- `birdNames` - list of chick names.
- `metaData (pd.DataFrame)` - meta details of all chicks (e.g. **sex**, **ripeRimu** etc).
- `birdWeights (pd.DataFrame)` - age and weights of chicks in sample.
- `optimalVariables (pd.DataFrame)` - top parameters for variables for all groups, such as the boundary scalar `k` for 19.
- `method` - Use `method=1.4` for best method.
- `HRInbetween` - replacement for those with **handReared** classified as "YN". Only used when `useHRBackUp==1`, else the training set will be based only on "YN" chicks.
- `useHRBackUp (bool)` - see above.
- `ripe` - **ripeRimu** filter, e.g. "Y" to filter for those born in Rimu season.
- `percentageScalar (np.float)` - percentage scalar for boundary in 19.
- `backupSex` - back-up sex if sex is unidentified. Only used if `useBackUp==1`.
- `useBackUp (bool)` - see above.
- `KarcherMeanDetails (pd.DataFrame)` - Karcher mean, gamma, and standard deviation details for all groups.

- `optVariables (list)` - optimal variables can be passed to use for all groups of chicks, in place of the `optimalVariables` parameter.

Note: for all other variables, please refer to the descriptions listed previously in 23. **`return results (dict)`** - all results: predicted health (*"predictedHealth"*), original weights (*"originalWeight"*), aligned weights (*"alignedWeight"*), Karcher mean for chick's group (*"KarcherMean"*), average gamma function for group (*"averageGamma"*), original weights of group (*"sampleOriginalWeights"*), aligned weights of group (*"sampleAlignedWeights"*), names of chicks in original training sample in group (*"sampleNames"*).

A.2 Code Examples

In order to visualise what the underlying code behind our Kākāpō tool (see 5.9) may look like, we provide an example of some Python code that can be used to predict the health of kākāpō chicks from their growth curves, which incorporate the functions outlined in the previous section.

```
# Imports
from  sourceFunctions.kakapomeans import predictHealth
import pandas as pd

# Load datasets:
meta = pd.read_csv('Data//Kakapo_Meta_Updated.csv')
all_weights = pd.read_csv('Data//All_Records_Filtered.csv')
Birdnames_ageKnown = np.unique(list(all_weights[~np.isnan(all_weights
    ['Age'])]['birdName']))
optVariables = pd.read_csv('Data//optimalVariables_Updated.csv',
    converters={'Variables': eval})

KMeanData = pd.read_csv('Data//kmeanDetails_allGroups.csv',converters
    ={'KarcherMean': eval,'standardDeviation': eval, 'averageGamma':
    eval})
# Choose chick and growth range:
name = "Kura"
minAge = 0
maxAge = 75

# Predict health:
results = predictHealth(name,Birdnames_ageKnown,meta,all_weights,
    optVariables,KarcherMeanDetails=KMeanData,minAge=minAge,maxAge=
    maxAge)

print(results['predictedHealth'])
```