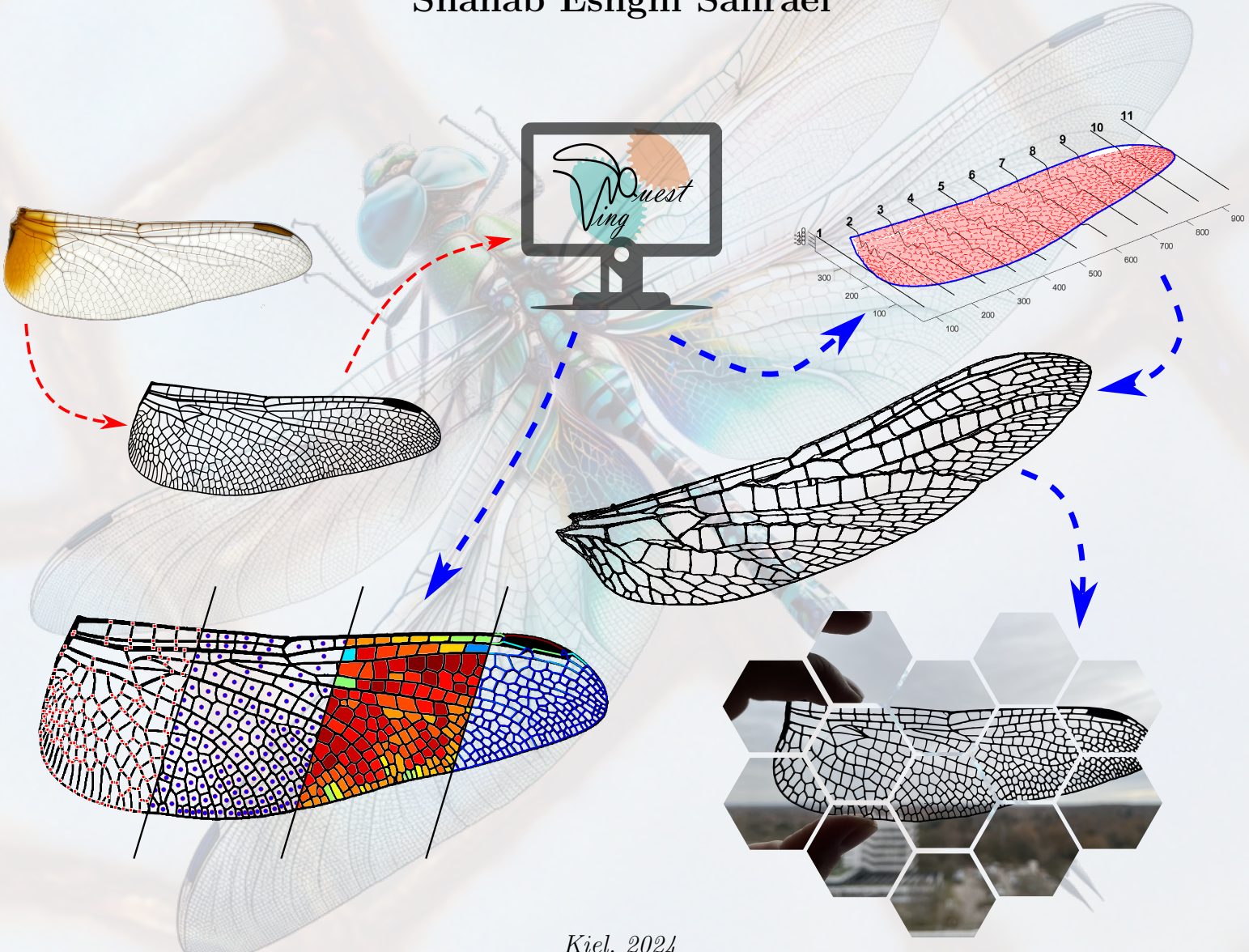
# Computer Vision-Based Analysis and 3D Modeling of Insect Wings

Dissertation
in fulfilment of the requirements for the degree of "Dr. -Ing"
of the Faculty of Mathematics and Natural Sciences
at Kiel University

submitted by

**Shahab Eshghi Sahraei**




*Kiel, 2024*

URN: urn:nbn:de:gbv:8:3-2024-00779-5

Printed and/or published with the support of the German Academic Exchange Service.

First Examiner: Prof. Dr. Stanislav N. Gorb
Second Examiner: Prof. Dr. Jan-Henning Dirks

Date of the Oral Examination: June 25, 2024

# Acknowledgements

This research was conducted between October 2019 and March 2024 in the Department of Functional Morphology and Biomechanics at the Zoological Institute of Christian-Albrechts-Universität zu Kiel (CAU), Germany. I would like to seize this moment to express gratitude to all those whose support played a crucial role in making this project feasible.

First and foremost, I extend my sincere gratitude to my supervisor Prof. Dr. Stanislav N. Gorb, the head of the Department of Functional Morphology and Biomechanics at Kiel University. His invaluable guidance, mentorship, and unwavering support have played a pivotal role in shaping my academic development.

I express my heartfelt gratitude to Dr. Hamed Rajabi (LSBU - London South Bank University), my advisor at London South Bank University, along with Dr. Vahid Nooraeefar (University of Guilan) and Dr. Kamyar Kasmaei (SLU - Swedish University of Agricultural Sciences) for their priceless guidance and advice. Their support played an essential role in obtaining the DAAD scholarship and paved the path for my journey to Germany.

I sincerely thank Prof. Dr. Jan-Henning Dirks for his invaluable mentorship and insights, which have significantly enriched my academic journey.

I appreciate the financial support from the German Academic Exchange Service (DAAD), which funded my doctoral studies through the 'Research Grants - Doctoral Programmes in Germany, 2019/20 (57440921)' for 54 months (01.10.2019 - 31.03.2024).

I am deeply grateful to Dr. Thies Büscher, for his insightful comments, valuable contributions to Paper IV: WingAnalogy, and assistance in translating the English abstract of this thesis into German, and to Dr. Natalia Matushkina (Taras Shevchenko National University of Kyiv) for her contributions to the same paper.

I thank my students, Johannes Poser, Lisa Claußen, Beta Ghifari, Hanna Ramien, Jana Ohlen, and Fiete Willmer, for their enthusiastic dedication and commitment.

I also thank my colleagues, both current and former, for their encouragement and invaluable feedback, particularly Mohsen Jafarpour, Dr. Ali Khaheshi, Dr. Elena Gorb, Dr. Wencke Krings, Dr. Chuchu Li, Dr. Alexander Kovalev, Esther Appel, Joachim Oesert, and Angela Veenendaal.

I want to express my deepest appreciation to my wife, Shamim, my parents, and my siblings Shadi, and Shahin for their unwavering support and understanding throughout my doctoral journey. Their love and dedication have been my constant source of strength and motivation.

I extend my deepest gratitude to my friends in Kiel and to all my friends whose unwavering support and encouragement have paved the way for my journey and studies in Germany. I especially thank Dr. Masoud Taleb for his invaluable advice and assistance.

Finally, I must express my deepest appreciation and remember my former supervisor and teacher at the University of Guilan, Professor Abolfazl Darvizeh (1951-2021). He not only guided me on the journey to becoming a researcher but also expanded my perspective by underscoring the importance of comprehending the universe through an engineering perspective. May his soul rest in peace.

# Abstract

The study of insect wings holds significant importance in both biology and engineering, offering insights into evolutionary adaptations and inspiring biomimetic designs that revolutionize aerodynamics and robotics. However, the complex geometry and the diversity of wing venation patterns among different species of insects, and even within one species, make their study challenging and time-consuming. This underscores the importance of developing automated techniques and methods.

In this thesis, the exploration of computer vision and image processing techniques in the study of insect wings has led to the development of four software packages. WingMesh extracts wing geometry and generates mesh using distmesh2d, while WingGram, developed for finite element (FE) modeling in Abaqus using Python scripting and computer vision, exhibits advantages in speed, model practicality, and applicability for intricate structures. Besides in WingMesh, and WingGram a new method is suggested for the assignment of corrugations effortlessly to the generated FE models using a secondary image. WingSegment, another developed software, segments insect wing images to extract cell boundaries, vein patterns, places of junctions, and the location of Pterostigma. It also measures morphological features such as the area, length, and circularity of cells, as well as the length of veins, and subsequently depicts related statistics in histograms, boxplots, and distribution contours. Moreover, it generates sketches of the wing in FC-Macro file format, which is compatible with FreeCAD for easy generation of 3D printable models. The last software, WingAnalogy, designed for automated asymmetry analysis, utilizes Particle Swarm Optimization (PSO) for automated superimposing and computer vision for extracting wing morphometry and computing the asymmetry. It has a tool, for dividing the wing, into cell sets, and measuring the asymmetry of local regions in the wing.

In this thesis, another critical focus lies in exploring the correlation between wing and cell size across 389 wings from various Odonata families. Despite diverse wing sizes, a remarkable constancy in cell size is observed. This consistency prompts the hypothesis that uniform cell size serves as an actual crack-stopping mechanism.

Evaluations and validations of the developed tools show their precision in modeling, morphometry, and asymmetry analysis applicable to the variety of insect wings. Developed methods are user-friendly, scalable, accessible, and accurate. They use embedded algorithms, and custom codes which makes them consistent, and objective. The methods and findings presented in this thesis collectively contribute to the advancement of computational entomology, providing efficient tools for the analysis of insect wings and yielding novel insights into their structural and functional characteristics.

# Kurzzusammenfassung

Die Untersuchung von Insektenflügeln ist sowohl in der Biologie als auch für den Transfer in die Ingenieurswissenschaften von großer Bedeutung. Sie bietet Einblicke in evolutionäre Anpassungen und inspiriert biomimetische Designs, die die Aerodynamik und Robotik revolutionieren. Die komplexe Geometrie und die Vielfalt der Flügeladerungsmustern bei verschiedenen Insektenarten und sogar innerhalb einer Art machen ihre Untersuchung jedoch schwierig und zeitaufwändig. Dies unterstreicht die Bedeutung der Entwicklung automatisierter Methoden.

In dieser Arbeit hat die Erforschung von Computer Vision und Bildverarbeitungstechniken bei der Untersuchung von Insektenflügeln zur Entwicklung von vier Softwarepaketen geführt. WingMesh extrahiert die Flügelgeometrie und erzeugt ein Netz mit distmesh2d. WingGram, das für die Finite-Elemente-Modellierung (FE) in Abaqus unter Verwendung von Python-Scripting und Computer Vision entwickelt wurde, hat darüber hinaus Vorteile in Bezug auf Geschwindigkeit, Modellierbarkeit und Anwendbarkeit für komplizierte Strukturen. Außerdem wird in WingMesh und WingGram eine neue Methode zur mühelosen Zuordnung von Oberflächenunebenheiten zu den generierten FE-Modellen unter Verwendung eines Sekundärbildes vorgeschlagen. WingSegment, eine weitere entwickelte Software, segmentiert Bilder von Insektenflügeln, um Zellgrenzen, Adermuster, Positionen von Verbindungen und die Lage von Pterostigmen zu extrahieren. Es misst auch morphologische Merkmale wie die Fläche, Länge und Zirkularität der Zellen sowie die Länge der Adern und stellt anschließend die entsprechenden Statistiken in Histogrammen, Boxplots und Verteilungskonturen dar. Darüber hinaus generiert es Skizzen des Flügels im FCMacro-Dateiformat, das mit FreeCAD kompatibel ist und die Erstellung von 3D-Druckmodellen erleichtert. Die letzte Software, WingAnalogy, wurde für die automatisierte Asymmetrieanalyse entwickelt und nutzt die Partikelschwarmoptimierung (PSO) für die automatisierte Überlagerung und die Computer Vision für die Extraktion der Flügelmorphometrie und die Berechnung der Asymmetrie. Es verfügt über ein Werkzeug zur Unterteilung des Flügels in Zellensätze und zur Messung der Asymmetrie lokaler Regionen des Flügels.

In dieser Arbeit liegt ein weiterer wichtiger Schwerpunkt auf der Untersuchung der Korrelation zwischen Flügel- und Zellgröße bei 389 Flügeln verschiedener Libellen-Familien. Trotz unterschiedlicher Flügelgrößen ist eine bemerkenswerte Konstanz der Zellgröße zu beobachten. Diese Konstanz legt die Hypothese nahe, dass eine einheitliche Zellgröße als intrinsischer Riss-Stopp-Mechanismus dient. Auswertungen und Validierungen der entwickelten Werkzeuge zeigen ihre Präzision bei der Modellierung, Morphometrie und Asymmetrieanalyse, die auf die Vielfalt der Insektenflügel anwendbar sind. Die entwickelten Methoden sind benutzerfreundlich, skalierbar, zugänglich und genau. Sie verwenden eingebettete Algorithmen und benutzerdefinierte Codes, was sie konsistent und objektiv macht. Die in dieser Arbeit vorgestellten Methoden und Ergebnisse tragen insgesamt zur Weiterentwicklung der computergestützten Entomologie bei, indem sie effiziente Werkzeuge für die Analyse von Insektenflügeln bereitstellen und neue Einblicke in deren strukturelle und funktionale Eigenschaften ermöglichen.

# Visual Abstract

## Developed Software Packages

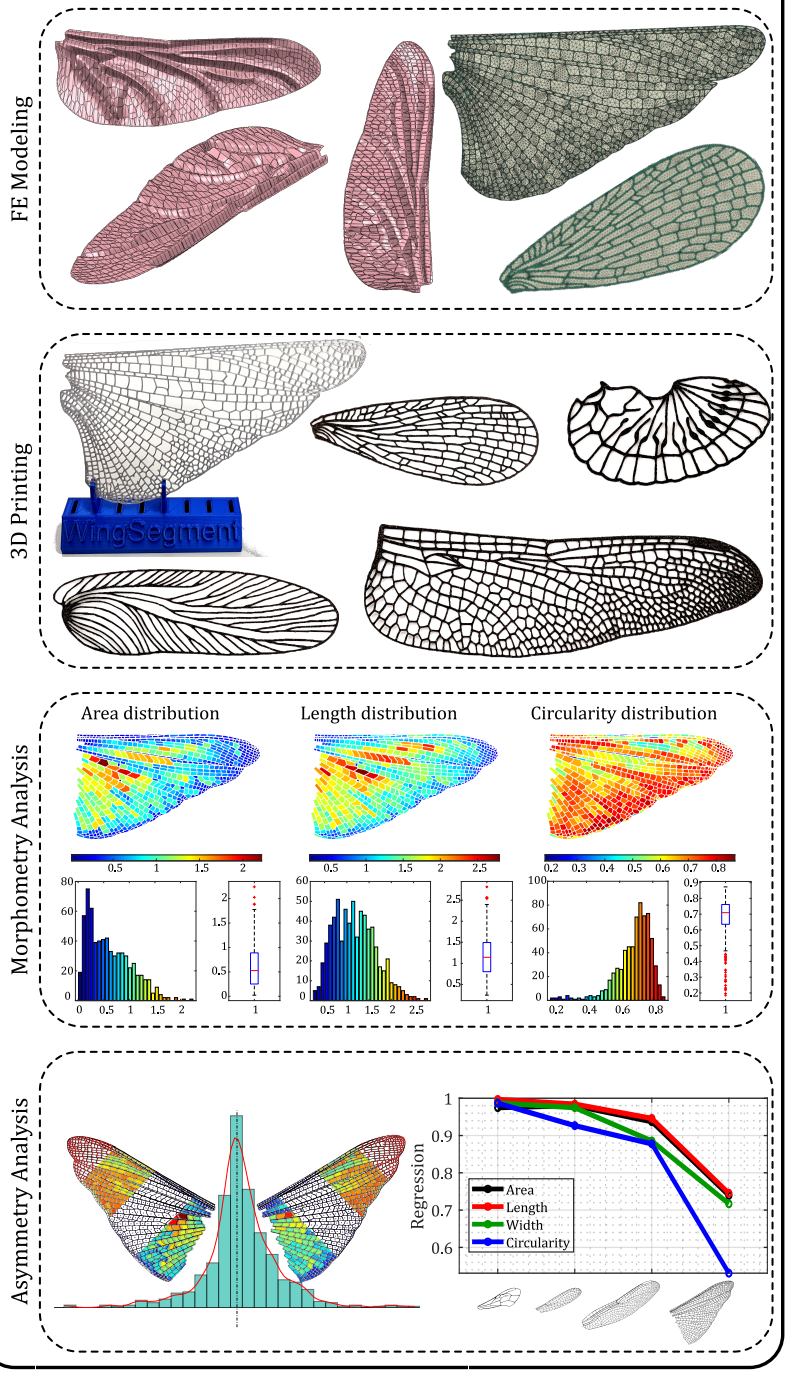
**Wing Analogy**
A software for computing the fluctuating asymmetry of insect wings

**Wing Mesh**
A mesh generation tool for insect wings, from insect wing image

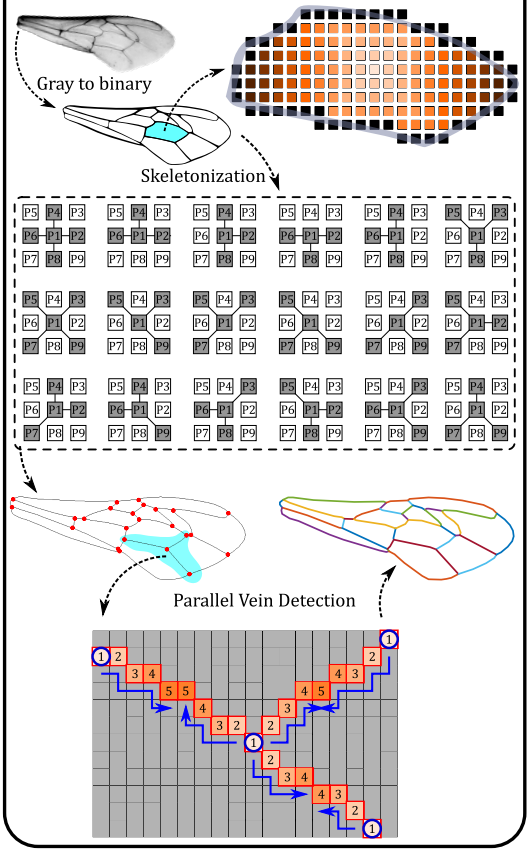
**Wing Gram**
A Matlab toolbox for automated Finite Element Modeling and geometrical analysis of insect wing

**Wing Segment**
A software for segmenting insect wing images, for extraction of junctions, cell boundaries, and vein patterns

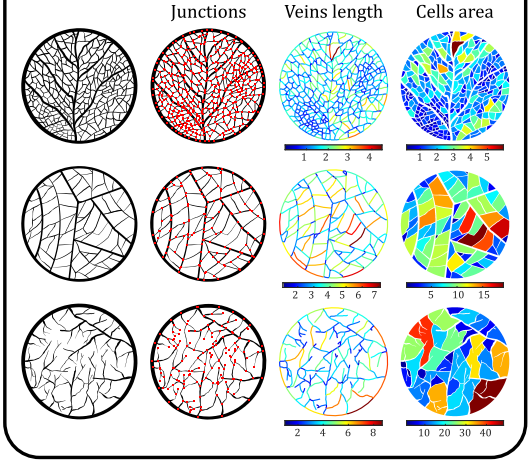
## Applications




## Computer Vision




## Beyond Entomology



# Contents

# List of Abbreviations

| | |
|---|---|
| CV | Computer Vision |
| DIP | Digital Image Processing |
| PSO | Particle Swarm Optimization |
| FE | Finite Element |
| FEM | Finite Element Modelling |
| FEA | Finite Element Analysis |
| FC | FreeCAD (Software) |
| GUI | Graphical User Interface |
| FDM | Fused Deposition Modeling |
| SVD | Sequential Vein Detection |
| PVD | Parallel Vein Detection |
| AI | Artificial Inteligence |
| ML | Machine Learning |
| DL | Deep Learning |
| NEFI | Network Extraction From Image (Software) |
| NET | Network Extraction Tool (Software) |
| CLSM | Confocal Laser Scanning Microscopy |
| APDL | Ansys Parametric Design Language |
| STL | Standard Triangle Language (File Format) |
| INP | Abaqus Input File |
| JNL | Abaqus Journal File |
| BA | Burning Algorithm |
| EXE | Executable (File Format) |
| PLA | Polylactic Acid |
| NRMSE | Normalized Root Mean Square Error |
| PDF | Portable Document Format (File Format) |
| TXT | Text (File Format) |
| CSV | Comma-Separated Values (File Format) |
| STD | Standard Deviation |
| CAE | Computer Aided Engineering |
| CAD | Computer Aided Design |

# Chapter 1

# Introduction

## 1.1 Insect Wings: Engineering Principles and Biological Function

In the intricate realm of entomology, the study of insect wings stands as a central point of exploration, providing an entrance to a multitude of scientific discoveries. The mechanical behavior of insect wings is a fascinating study of how nature combines materials, structure, and design to achieve highly efficient and adaptive mechanisms for flight [1]. As depicted in Figure 1.1 insect wings are essentially composite structures made of veins and membranes, each contributing to their overall mechanical performance [2, 3]. The veins, made of a rigid material such as chitin, are the main load-bearing elements that provide structural support and resist bending and deformation during flight [4]. They are carefully positioned to maximize the wing's strength while minimizing its weight, which is known as the spatially varying stiffness strategy [5]. This arrangement enables wings to be strong yet flexible, which allows them to sustain the aerodynamic forces associated with rapid maneuvers and landing and takeoff.

The thin membrane regions between the rigid veins of the insect wing play a critical role in its ability to take flight, enabling it to generate both lift[1] and thrust[2] [6]. These thin, flexible, and lightweight membranes allow the wing to change shape during flight. The elastic nature of the membrane complements the veins' rigidity, resulting in a wing that can deform radically without structural failure [7, 8]. This interaction between veins and membranes is essential to the wing's aerodynamic performance, as it allows the fine-tuning of wing shape and angle to control flight dynamics [7]. In addition, the junctions where veins meet or intersect are of both mechanical and aerodynamic interest. These have been designed such that stresses experienced by the wing are redistributed throughout its structure, ensuring that no single point is subjected to such loads that could lead to structural failure [9, 10].

The corrugations found across the wing surface add another layer of complexity to the mechanical behavior of wings (Figure 1.1) [11–14]. They increase the bending stiffness of the wing. This is essential for resisting the enormous dynamic aerodynamic forces encountered in flight as well as the smaller static forces which, without restriction on the wing's bending, would quickly lead to the wing folding or collapsing. [11, 13]. The corrugated structure also reduces the energy cost of flight, thus optimizing aerodynamic efficiency[11]. The intricate combination of veins, membranes, junctions, and corrugations

[1]Lift is the force that opposes the force of gravity, allowing an insect to stay airborne.
[2]Thrust is the force that moves insect forward, acting parallel to the direction of motion.

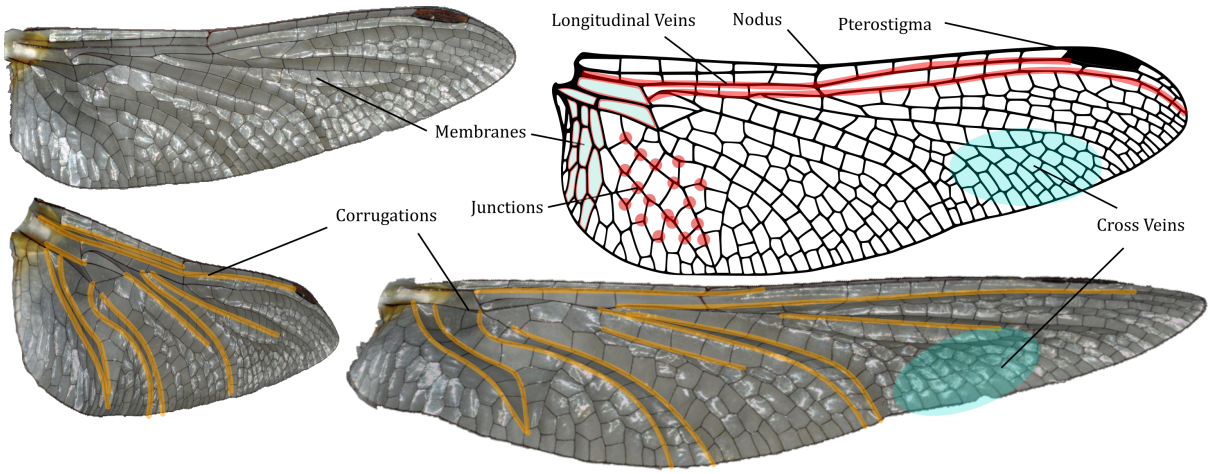



Figure 1.1: The components of a dragonfly hindwing

in insect wings provides an impressive example of a natural engineering solution, where mechanical behavior is optimized for survival in a wide range of aerial maneuvers and environmental conditions.

More than just tools for flying, insect wings offer important information from different angles, going beyond functional morphology and biomechanics. The areas listed below represent some of the most important subjects in which the study of insect wings serves as a significant focus. i) Taxonomy and Identification. Wing vein patterns are like unique fingerprints, helping scientists and researchers identify and classify different insect species [15–18]. ii) Evolutionary Insights. By studying how wings have changed over time, scientists can uncover the history of insects, understanding how different species are related and how they've evolved [19, 20]. iii) Functional Morphology and Biomechanics. Insect wings are a source of inspiration for engineers in the design and development of new tools, materials, and structures such as artificial wings for flapping robots [21], attachment systems [22–24], bioinspired joints [25, 26]. iv) Environmental Indicators. The examination of changes in insect wing morphology and asymmetry can serve as early indicators of environmental shifts, including those by climate change. Monitoring these changes contributes to our understanding of the ecological impact of environmental transformations [27–29]. v) Pest Management. In the agricultural landscape, the study of insect wings emerges as a vital tool for pest management. Identification of pest species through wing characteristics enables the development of targeted strategies to mitigate the impact of these insects on crops and ecosystems [30]. vi) Paleontology. The study of insect wings is vital in paleontology as fossilized wings provide crucial evidence for understanding ancient ecosystems, offering insights into prehistoric insect diversity and environmental conditions [31, 32].

## 1.2 Challenges in Insect Wing Research

### 1.2.1 Experimental Study and Finite Element Modeling

Wings are tiny, fragile tissues that are highly sensitive to external forces, rendering studying their mechanical behavior particularly challenging. This is due to the necessity for specialized sensors and machinery. This highlights the importance of using numerical solutions like the finite element method (FEM) in this field. In recent years many FEM-based studies have been conducted on insect wing models [2, 6, 13, 33–35]. FEM is a

computational technique used to approximate the solutions of boundary value problems in engineering and physics. It is particularly useful for problems with complex geometries, material properties, and boundary conditions. FEM divides a large problem into smaller, simpler parts called finite elements. Apart from the difficulties in experimental tests, in the realm of insect wings, FEM is essential for several reasons. i) Biomechanics and Structural Analysis. FEM allows researchers to simulate and analyze the structural behavior of wings under various conditions to study flexibility, deformation, and response to different types of loads. This helps in understanding how the wings withstand aerodynamic forces, vibrations, and other mechanical stresses [2, 13, 33, 34]. ii) Optimization. FEM allows researchers to optimize the design of insect wings for specific functions, such as efficient flight or maneuverability. By simulating different wing structures and materials, scientists can identify the most effective configurations for achieving optimal aerodynamic performance [36–38]. iii) Computer simulations offer extra information compared to experimental tests. In computer simulations, not only various loading scenarios can be conducted, but they also enable comprehensive visualizations of the mechanical behavior of the structure, such as depicting stress or displacement distributions. Additionally, in FEM, the model can be manipulated to conduct systematic studies, e.g., by excluding certain elements like veins or corrugations, which is not possible in real experiments. Furthermore, FEM simulations are scalable and reproducible with minimal effort [2, 14, 33, 39].

Generating a precise model of the structure is essential before conducting the FE analysis. Insect wings, with their complex geometry, present a unique set of challenges in FE modeling [39, 40]. Creating 3D models of insect wings is a challenging task due to their intricate geometry, resulting in a time-consuming and error-prone process. This complexity often leads to a need for simplification in the 3D modeling of insect wings. To speed up the modeling process, many Finite Element (FE) models depict insect wings as 2D planar structures, even though insect wings possess three-dimensional characteristics [35, 39, 41].

### 1.2.2 Morphometry and Asymmetry Analysis

In the study of the morphology of insect wings, the amount and place of junctions and cells, the size, and shape of cells, and the venation pattern are essential. Extracting these features from insect wings, with their complex geometry, presents a unique set of challenges for researchers [42, 43]. Insects with larger wings, such as dragonflies, damselflies, stick insects, and grasshoppers, tend to have more intricate vein patterns [44]. Extracting these features manually is very time-consuming, error-prone, and inaccurate. In addition, examining wing asymmetry[3] becomes even more challenging when dealing with complex wing structures [45]. Insects naturally exhibit variations in wing size and shape, making it tricky to distinguish if observed asymmetry is significant or just individual differences. Traditional measurement methods including landmarking, and manual cell counting or using micro-CT scanners may struggle to capture these details accurately [46–54]. In conclusion, exploring the study of insect wing morphology proves to be a complex undertaking that demands specialized tools, time, precision, and skilled researchers. However, manual analysis and investigations are still not efficient, this underscores the necessity for innovative, automated, and precise methodologies, leveraging advancements in tools and technologies such as computer vision.

[3]Asymmetry in insect wings refers to differences in size, shape, structure, or pattern between the left and right wings of an individual insect

## 1.3 Computer Vision and Digital Image Processing in the Study of Insect Wings

In recent years, there has been a development in computer tools and studies, employing computer vision[4] and image-processing[5] techniques to investigate the intricacies of insect wings. These methods primarily focus on extracting geometric features from wing images [55–60]. ImageJ is a well-known software package for analyzing biological structures, offering a range of tools and filters that enhance its effectiveness in studying biological structures, including insect wings [61]. Despite its versatility, some tools within ImageJ are automated, while others are not. It is important to note that ImageJ, in general, is not specifically tailored for the study of insect wings. However, an extension to ImageJ called FijiWings has gained recognition for its specialization in studying *Drosophila* [57]. This extension is designed for measuring cell numbers and tissue size in *Drosophila* wings. There are also other studies presenting automated or semiautomated methods for the study of *Drosophila* wings [62–64]. DrawWing is a dedicated software developed for the analysis of honeybee wings (*Apis mellifera*). This specialized tool is designed to automatically extract the locations of wing junctions. It is important to note that DrawWing requires high-resolution images, specifically those with a resolution exceeding 2400 dpi[6] in both dimensions (2400 dpi × 2400 dpi) [56]. In addition to DrawWing, there are other software packages designed for the extraction of network patterns from insect wings. NEFI (Network Extraction From Image) and NET (Network Extraction Tool) are two notable software tools specifically developed for this purpose [65, 66]. Nonetheless, these tools focus on extracting the venation network from insect wing images, but they face a limitation in that they cannot precisely extract the path of the venation. In other words, if there is a curved vein between two junctions in a wing, these methods can only identify the connection of those two junctions through the curved vein and cannot capture the exact trajectory of that vein. Moreover, several other studies have explored the integration of machine learning and computer vision techniques in the study of insect wings [55, 67–70].

### 1.3.1 Prior Self-Contribution to the Field

Before this thesis, I undertook contributions to the development of new methodologies and software packages based on image processing and computer vision for the study of biological structures, especially insect wings.

#### Automated Finite Element Modeling of Insect Wings

In 2016, a simple method for finite element modeling of insect wings was developed [58]. This method was designed to detect contrast between white and black pixels, distinguishing membranes (white pixels) from veins (black pixels). The code then filled the entire wing area with small squares, serving as meshes. These squares, considered as the mesh elements, maintained a uniform size throughout the entire wing, although the size could be initially adjusted by the user. Additionally, the code categorized the squares into

[4]Computer vision is the technology that allows computers to interpret and understand visual information from the world, similarly to human vision.

[5]Image processing involves manipulating and analyzing digital images to improve their quality, extract information, or apply visual effects, using algorithms and computer-based techniques.

[6]DPI stands for "Dots Per Inch," a measurement used to describe the resolution of a printer, scanner, or a digital image's resolution.

two groups based on the color of veins and membranes. Finally, the coordinates of the squares were stored using APDL (Ansys parametric design language) to create a finite element model of the wing. Notably, this computer tool introduced a novel method for assigning corrugations to the wing model for the first time, utilizing a secondary image. Subsequently, this method was further refined by integrating the region growing method [71, 72] to differentiate each wing cell as a separate section. Wing models generated through this method have been employed in multiple studies to investigate the mechanical behavior of insect wings [2, 13, 34]. However, limitations in controlling the mesh size, and geometry were the most disadvantage of these methods.

#### Computing the Probability of Failure in Insect Wings

Another developed method based on computer vision computed the probability of failure in insect wings. This methodology used several damaged wing images and, by investigating damaged areas, presented a heatmap regarding the intensity of failure in different regions of the wing. This method was used for investigating the probability of wing damage in migrating butterflies and in the dragonfly *Sympetrum vulgatum* (Anisoptera: Libellulidae) [73, 74].

#### Eestimator: A Software for Analysing CLSM Images

Eestimator (depicted in Figure S1) is a software that processes images taken by a Confocal Laser Scanning Microscope (CLSM) for determining the spatial gradient of the elastic modulus of insect cuticle [75].

#### Analysis of the Significance of Insect Cuticle Using Multi-Objective Optimization

This study integrated finite element analysis with multi-objective optimization to determine the functional significance of graded properties in insect cuticles. My role in this research involved developing the MATLAB code for multi-objective optimization and establishing the interface between MATLAB and Abaqus CAE to automate the finite element analysis of models developed through the optimization method [76].

## 1.4 Research Questions

Considering the difficulties in the study of insect wings this thesis explores the following questions:

I. How can image processing (IP) and computer vision (CV) techniques be developed and applied for the automated segmentation and morphology analysis of insect wings?

II. What methods can be established for automated finite element (FE) modeling and 3D printing of insect wings from their images, and what does it reveal about the structural and functional aspects of insect wings?

III. Which IP and CV strategy can contribute to the easy assignment of corrugations in FE modeling of insect wings?

IV. How IP and CV can automate asymmetry analysis of insect wings to offer more reliable, scalable, fast, and precise evaluations?

## 1.5 Thesis Objectives

To address the research questions in the study of insect wings, this thesis aims to achieve the following objectives.

I. **Method Development for Automated Segmentation of Insect Wing Images.** This objective explores new methods and algorithms for segmenting the wing image to extract the boundary of the wing outline, wing cells, the place of junctions, and the vein patterns in the wing.

II. **Method Development for Automated Morphometry Analysis of Insect Wings.** This objective investigates methods for generating statistical data on the wing, including measurements such as area, length, width, and circularity of wing cells, as well as the length of veins.

III. **Method Development for Automated 3D Modeling of Insect Wings.** This objective investigates the use of extracted geometries from wing images to automatically generate finite element (FE) models of insect wings. This involves developing new methods for the seamless assignment of corrugations to the wings. Additionally, it explores the process of generating 3D models suitable for the 3D printing of insect wings.

IV. **Method Development for Asymmetry Analysis of Insect Wing Pairs.** This objective explores solutions for the automated asymmetry analysis of insect wings. These solutions encompass automated wing superimposition, the examination of relevant statistics, and the establishment of criteria for asymmetry computation. This approach aims to calculate asymmetry based on features automatically extracted from the wing image.

V. **Evaluation of Methodological Effectiveness.** This objective seeks to perform comprehensive evaluations of the developed methods, comparing them with traditional manual and semi-automated techniques. The aim is to determine enhancements in accuracy, efficiency, and applicability for a broad spectrum of insect species and wing types.

VI. **Exploration of Biological Insights.** This objective aims to examine the functional morphology of insect wing cellularity across different species, focusing on how cell size varies across different parts of the wing as the overall wing size changes.

## 1.6 Scholarly Outputs

The following manuscripts, either published, accepted, or currently under consideration, derive from the findings of this doctoral thesis.

I. S Eshghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. WingMesh: A MATLAB-based application for finite element modeling of insect wings. *Insects*, 11(8):546, 2020 [59]. https://doi.org/10.3390/insects11080546.

II. S Eshghi, F Nabati, S Shafaghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. An image based application in MATLAB for automated modelling and morphological analysis of insect wings. *Scientific Reports*, 12(1):13917, 2022 [60].

https://doi.org/10.1038/s41598-022-17859-9.

III. S Eshghi, H Rajabi, J Poser, and SN Gorb. WingSegment: A Computer Vision-Based Hybrid Approach for Insect Wing Image Segmentation and 3D Printing. *Advanced Intelligent Systems*, 6(5):2300712, 2024 [77].

https://doi.org/10.1002/aisy.202300712.

IV. S Eshghi, H Rajabi, N Matushkina, L Claußen, J Poser, TH Büscher, and SN Gorb. WingAnalogy: A Computer Vision Based Tool for Automated Insect Wing Asymmetry and Morphometry Analysis. *Under review.*

V. S Eshghi, H Rajabi, F Nabati, S Shafaghi, S Nazerian, A Darvizeh, and SN Gorb. Allometric scaling reveals evolutionary constraint on Odonata wing cellularity via critical crack length. *Advanced Science*, 11(23):2400844, 2024 [78].

https://doi.org/10.1002/advs.202400844.

## 1.7 Overview of the Thesis Structure

This thesis is divided into ten chapters.

- **Chapter 1 (Introduction)** provides an introduction to insect wings research and discusses their significance in both engineering and biology. A literature review of previous works is included, along with research questions and objectives of the thesis. Additionally, a list of published papers from this thesis is provided.
- **Chapter 2 (Methodology Overview)** delves into the methodologies used throughout this research and those applied in this thesis.

Subsequently, five papers forming the basis of the thesis are presented in Chapters 3-7.

- **Chapter 3 (Paper I: WingMesh)** introduces WingMesh [59], a new approach for generating meshes on insect wing domains using computer vision and distmesh2d.
- **Chapter 4 (Paper II: WingGram)** presents WingGram [60], another tool designed for the 3D modeling of insect wings through computer vision and Python scripting in Abaqus.
- **Chapter 5 (Paper III: WingSegment)** explores WingSegment,[77] a software developed for segmenting insect wing images. WingSegment introduces a new hybrid method for extracting venation patterns and utilizes FCMacro files to automatically generate 3D models for FreeCAD software, advancing the creation of 3D models for 3D printing of insect wings.
- **Chapter 6 (Paper IV: WingAnalogy)** introduces a software package for automated asymmetry analysis of insect wings, equipped with tools for automated measurement and extraction of wing morphometry properties. This software incorporates a Particle Swarm Optimization method for the automated superimposition of wings.
- **Chapter 7 (Paper V: Allometric Scaling on Odonata Wings)** investigates the relationship between wing size and wing cell size in Odonata wings from different species with varying wing sizes.

Afterwards the thesis continues with discussion, ongoing projects and future works, and conclusion.

- **Chapter 8 (General Discussion)** discusses the application of the developed methods, and their limitations, and compares them with other tools developed in the literature.
- **Chapter 9 (Ongoing Projects and Future Directions)** describes ongoing projects, and future directions.
- **Chapter 10 (Conclusion)** provides conclusions of the thesis.

This thesis includes Appendices following the references. In the Appendices, all primary codes developed in this study are documented. Moreover, several supplementary figures and tables are documented in the Appendices. Additionally, supplementary videos and files are stored in a Zenodo repository, with their descriptions accessible in the Appendices.

# Chapter 2

# Methodology Overview

## 2.1 Digital Image Processing

In recent years, digital image processing has emerged as a powerful tool for analyzing and extracting valuable information from various images. In this doctoral thesis, the application of digital image processing techniques to analyze insect wing images is explained. By converting these images into binary representations the veins and membranes can effectively be differentiated. This introductory section provides an overview of digital image processing, the concept of digital images and pixels, the different color models such as RGB, the significance of grayscale and binary images, and the role of thresholding in image segmentation.

### 2.1.1 Digital Images and Pixels

Digital images are representations of visual data in a discrete form, consisting of a collection of picture elements or pixels. Each pixel corresponds to a small image area and carries information about its color or intensity. Pixels are the building blocks of digital images, forming a grid-like structure where combining their values creates the overall visual representation. The resolution of an image is determined by the number of pixels, with higher resolutions offering more detail and clarity.

#### RGB Images

RGB (Red, Green, Blue) is a widely used color model in digital image processing. In RGB images, each pixel is represented by three color channels: red, green, and blue. The intensity values of these channels range from 0 to 255, representing the varying degrees of red, green, and blue. Therefore, RGB images can represent up to 16,777,216 different colors, enabling the reproduction of lifelike and vivid visualizations.

#### Grayscale Images

Grayscale images represent the intensity or brightness of pixels without color information. Each pixel in a grayscale image is assigned an intensity value ranging from 0 (black) to 255 (white), often represented as an 8-bit integer. Grayscale images simplify the image to shades of gray, offering simplicity and objectivity that are particularly useful in scientific studies and image analysis tasks.

#### Black and White (Binary) Images

Binary images, on the other hand, have only two pixel values: 0 and 1. These values represent black and white, respectively. Binary images simplify the image to a stark contrast between black and white, making it easier to distinguish structures and patterns. The binary representation is especially advantageous in scientific studies, computer vision applications, and image segmentation tasks due to its simplicity and computational efficiency.

Black and white, i.e., binary images, offer several advantages in scientific studies and image analysis. The high contrast between black and white allows for easy identification of important structures and features. By eliminating color information, black-and-white images emphasize key elements and reduce distractions, enabling researchers to focus on specific details and patterns. Furthermore, the simplicity and computational efficiency of binary images make them well-suited for various image-processing algorithms and analytical techniques.

### 2.1.2 Thresholding and Conversion to Binary Images

Thresholding plays a vital role in converting digital images to binary representations. It involves selecting a threshold value that separates black pixels from white pixels. Pixels with intensity values below the threshold are assigned a value of 0 (black), while those above or equal to the threshold are assigned a value of 1 (white). Various thresholding algorithms, such as global thresholding, adaptive thresholding, and Otsu's method, can be employed to automate the selection of an optimal threshold value based on image characteristics and the desired segmentation outcome. By applying thresholding algorithms to RGB or grayscale images of insect wings, they can be transformed into binary representations that enhance the contrast between veins and membranes.

### 2.1.3 Preparing Image

Images for developed methods can be taken using cameras or scanners however human intervention in preparing images for all developed tools is crucial as it significantly impacts their efficiency. This task, while important, is manageable. An input image requires the following specifications. i) Veins must appear black, and membranes white. ii) The wing's outline must be complete, ensuring it is entirely separated from its surroundings. iii) Very large images may increase runtime. A resolution of 300 dpi is recommended for efficient processing. iv) For optimal results, wings should be fully unfolded before scanning. This can be achieved by applying slight pressure during the scanning process. Figure 4.2m-q shows well-prepared and not suitable images as inputs for the tools in this study.

## 2.2 Computer Vision

Computer vision is a rapidly evolving field that focuses on enabling computers to interpret and understand visual information from images or videos. It involves the development of algorithms and techniques to extract meaningful information from visual data, mimicking the human visual system's ability to perceive and comprehend the world. With its wide range of applications in various domains, computer vision has become an indispensable tool in scientific research, industry, and everyday life.

One of the fundamental tasks in computer vision is image segmentation, which involves partitioning an image into distinct regions or objects. Region growing is a popular

technique used for image segmentation, particularly in cases where the regions of interest exhibit homogeneous characteristics. By iteratively expanding seed points or regions based on predefined criteria, region growing methods can effectively delineate boundaries and separate objects within an image.

### 2.2.1 Region Growing Method for Image Segmentation

Region growing is a widely used image segmentation technique in computer vision that aims to partition an image into coherent regions based on some predefined criteria [71, 72, 79, 80]. The core idea behind region growing is to iteratively expand seed points or regions by incorporating neighboring pixels that satisfy a certain similarity condition. The process continues until the entire image is segmented into distinct regions. Code S3 is an outline of the region growing technique that is applied in MATLAB to perform image segmentation of insect wings as part of this doctoral research.

### 2.2.2 Zhang-Suen Thinning Algorithm

In this Thesis, the Zhang-Suen thinning algorithm (i.e. skeletonization method) is incorporated as another method for segmenting insect wing images [81]. The Zhang-Suen algorithm is a well-established technique utilized for extracting the skeletal structure from binary images. By performing a series of sequential sub-iterations, the algorithm progressively thins the foreground pixels while preserving the connectivity and topology of the original shape. Each sub-iteration is designed to eliminate specific types of pixels, resulting in an accurate representation of the skeletonized structure. Code S22 in the Appendices is the custom code developed for this method.

### 2.2.3 Detection of Vein Junctions

The conditions applied in the skeletonized image to detect the locations of wing junctions in section 5.2.3 are thoroughly discussed. Each condition is explained, providing a comprehensive understanding of the criteria used for identifying these critical points in the wing structure. Code S5 in the Appendices is the developed code in MATLAB for junction detection.

## 2.3 Graph Theory

Graph theory is a mathematical discipline that studies graphs, which consist of vertices connected by edges. In this thesis, two graph theory methods i.e. line simplification and path-finding, are employed to analyze insect wing structures. Line simplification reduces the complexity of wing boundaries by removing unnecessary points while preserving essential features. Path-finding algorithms extract venation patterns by determining optimal paths between wing junctions.

### 2.3.1 Path-Finding Method

In this thesis, an improved path-finding method is developed to extract the venation pattern from insect wing images. This method is primarily based on the well-known Dijkstra algorithm [82], which is commonly used for solving maze problems and finding the shortest distance between two points. After detecting the locations of junctions

within the wing structure, a new strategy was devised considering the fact that at least one path was connected to each junction. This approach enabled us to trace the paths and determine the connectivity of the veins, ultimately revealing the complete venation pattern. In section 5.2.4 (Vein Path Extraction), a comprehensive explanation of the path-finding method based on the Dijkstra algorithm and how it is tailored to suit the task of extracting the venation pattern of insect wings is outlined. Corresponding codes for vein detection are documented and described as Code S6 and Code S7 in the Appendices.

### 2.3.2 Line Simplification

The aforementioned techniques utilized for extracting the boundaries of cells and venation patterns result in the detection of numerous pixels along these boundaries. However, many of these points are deemed unnecessary, and their removal does not significantly impact the geometric properties of the detected objects. On the other hand, retaining all these points leads to increased runtime and a larger volume of data. To address this issue, the Ramer-Douglas-Peucker line simplification method is described in Section 4.2.2 (Ramer-Douglas-Peucker Line Simplification Algorithm) [83, 84]. This method reduces the number of points along the boundaries while removing unnecessary ones. Applying this technique, effectively streamlines the representation of the boundaries, optimizing computational efficiency and reducing the overall data size. The developed code for the line simplification method is documented as Code S13 in the Appendices.

## 2.4 App Development in MATLAB

App Designer in MATLAB is utilized to develop customized software packages for this thesis. These software packages, namely WingMesh [59] (Chapter 3), WingGram [60] (Chapter 4), WingSegment [77] (Chapter 5), and WingAnalogy (Chapter 6) were specifically designed to address various aspects of the study of insect wings.

App Designer is a powerful tool in MATLAB that enables the creation of interactive and user-friendly graphical user interfaces (GUIs) for data analysis, visualization, and application development. It offers a drag-and-drop interface for designing the layout of the GUI, along with the ability to incorporate different components such as buttons, sliders, plots, and text fields.

## 2.5 Python Scripting in Abaqus

Abaqus is a widely used software platform for finite element analysis, offering a comprehensive suite of tools for simulating complex mechanical behavior. By integrating Python scripting into the Abaqus workflow, one can streamline the process of creating finite element models based on a desired geometry (i.e. extracted wing geometry). Python, being a versatile programming language with extensive libraries and modules, provided the necessary flexibility and functionality for tasks such as data processing, numerical computation, and automation. Python scripting is utilized to define wing geometry, generate mesh structures, assign corrugation, define different sections, and assign material properties. By incorporating Python scripting within the Abaqus environment, the generation of finite element models in the format of JNL for the insect wings, utilizing the extracted geometry successfully is automated. More information is provided in section

4.2.4. Besides, Developed codes for the generation of JNL files are documented as Code S16, Code S17, Code S18, Code S19 in the Appendices.

## 2.6 FreeCAD Macro

FreeCAD is an open-source parametric 3D modeling software that provides a robust platform for creating complex geometric models. The Macro files contain a series of predefined instructions and commands that can be executed within FreeCAD, allowing for the automatic generation of wing models with specific dimensions, features, and structural components. The integration of computer vision techniques with FreeCAD Macro file generation offered a seamless workflow for transforming the extracted wing geometry into accurate and customizable 3D models in FreeCAD. More description is available in Section 2.6 (FreeCAD Macro). The developed code for generating the FreeCAD Macro file is documented as Code S8 in the Appendices.

## 2.7 3D Printing

WingMesh [59], WingGram [60], and WingSegment [77] each generate INP, JNL, and FCMacro files, respectively. INP and JNL formats are compatible with Abaqus, while FCMacro is compatible with FreeCAD. The 3D models generated can be imported into Abaqus or FreeCAD and subsequently exported as 3D print-compatible files, specifically STL files. In this thesis, wing models are 3D printed using Fused Deposition Modeling (FDM) 3D printers, with a focus on a Prusa MK3S printer equipped with a 0.25 mm nozzle. The use of the 0.25 mm nozzle allows for more detailed prints and facilitates the creation of smaller wings.

# Chapter 3

# *Paper I.* WingMesh: A MATLAB-based application for finite element modeling of insect wings.[1]


## Summary

The finite element (FE) method is one of the most widely used numerical techniques for the simulation of the mechanical behavior of engineering and biological objects. Although very efficient, the use of the FE method relies on the development of accurate models of the objects under consideration. The development of detailed FE models of often complex-shaped objects, however, can be a time-consuming and error-prone procedure in practice. Hence, many researchers aim to reach a compromise between the simplicity and accuracy of their developed models. In this study, we adapted distmesh2d, a popular meshing tool, to develop a powerful application for the modeling of geometrically complex objects, such as insect wings. The use of the burning algorithm (BA) in digital image processing (DIP) enabled our method to automatically detect an arbitrary domain and its subdomains in a given image. This algorithm, in combination with the mesh generator distmesh2d, was used to develop detailed FE models of both planar and out-of-plane (i.e., three-dimensionally corrugated) domains containing discontinuities consisting of numerous subdomains. To easily implement the method, we developed an application using the MATLAB App Designer. This application, called WingMesh, was particularly designed and applied for rapid numerical modeling of complicated insect wings but is also applicable for modeling purposes in the earth, engineering, mathematical, and physical sciences.



[1]This chapter has been published in *Insects (MDPI)* under an open access license on 18 August 2020. Reproduced with permission from MDPI.
Citation: S Eshghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. WingMesh: A MATLAB-based application for finite element modeling of insect wings. *Insects*, 11(8):546, 2020.
https://doi.org/10.3390/insects11080546.

## 3.1 Introduction

The finite element (FE) method is a numerical technique that is generally used to simulate a physical phenomenon in the virtual world by solving complex boundary value problems [85]. FE software packages were developed to simplify often complicated simulation processes. They are especially very common in engineering applications [86–88] and are becoming increasingly popular in the investigation of the mechanical behavior of biological structures, such as complex human and animal body parts [1, 89–95].

Although providing a user with a high degree of flexibility, all available FE packages have a common need: an accurate model. A model is a domain that is subdivided into smaller polygonal or polyhedral meshes, so-called elements [96]. A modeling process, however, may present many challenges and can be rather time-consuming, especially when dealing with complex geometries [39, 97, 98], which is usually the case in biology. The skills of the software user can also strongly influence the process and the final result. These often lead to oversimplified models and, therefore, can affect the accuracy of simulation results. How can this problem be overcome?

In 2004, Persson and Strang aimed to address this problem by developing a simple meshing technique called distmesh2d [99]. As intended by its developers, the method, which was implemented in MATLAB code, provided an effective tool to mesh a given domain automatically. The simplicity of the method and the high quality of the produced mesh are the key advantages of the proposed method. However, it also has a major drawback: finding the distance to boundaries by the use of the mathematical equation f(x, y) = 0 or by values of a discrete set of points, as explained by the authors, is a time-consuming and error-prone procedure for complex geometries. Due to the use of a mathematical scheme to define the distance function, distmesh2d also has limitations when meshing a domain containing discontinuities.

Here, we aimed to address these challenges and improve the performance of distmesh2d but still maintain its simplicity. To this end, we used computer vision to automatically detect the boundary of a domain in a given image. We combined it with the mesh generator distmesh2d to develop an application for the rapid modeling of geometrically complex domains that consist of several subdomains. The applicability of the method is not limited to in-plane domains, but it can also mesh out-of-plane (i.e., corrugated) objects. We specifically designed and used our method to develop models of insect wings. The proposed application, called WingMesh, draws extensively on Persson and Strang's account in an attempt to offer a simple but more practical meshing tool.

## 3.2 Material and Methods

The modeling method presented in this study, called WingMesh, consists of several algorithms that interact with each other. The method requires an input image to identify the boundaries of a given domain as well as subdomains within that. The code distmesh2d is then employed to mesh the identified domain. Other algorithms were added to the main algorithms to model out-of-plane domains and create an INP file, which is the Abaqus input file format. Code S1 is the developed MATLAB code for WingMesh.

### 3.2.1 Burning Algorithm for Detection of the Boundary of a Given Domain

The burning algorithm (BA) was used to extract the boundary of a domain in a given image [71, 72]. The BA needs a digital black and white image as the input, in which black pixels, with a pixel value of 0, represent the domain's border, and white pixels, with a pixel value of 1, represent regions situated inside and outside of the domain. The BA uses the input image's matrix to detect the domain's boundary in that image. This process starts with choosing a pixel within the domain by the user (Pixel 1 in Figure 3.1a) and continues by detecting white pixels around the selected pixel. To this end, the algorithm checks the colors of pixels located in the four orthogonal directions of the selected white pixels (Pixels 2 in Figure 3.1b). The coordinates of the found black pixels are stored in a matrix, and the colors of detected white and black pixels are changed to 0.8 (light grey) and 0.1 (dark grey), respectively, in order to avoid their reselection in the next iteration. This process continues by searching for white and black pixels around only white pixels detected in the previous iteration (Figure 3.1c–m). This process continues until all white pixels inside the domain are detected (Figure 3.1m). Code S21 and *WingMesh Video S1* in the Supplementary Data are the source code of BA and a simple illustration of how it works, respectively [100].

### 3.2.2 Detection of Subdomains Within a Given Domain

The function BA can also detect subdomains within a given domain. When the main domain contains any subdomain, the application first finds the white pixels outside the domain. This process eventually results in the detection of the boundary of the domain.

To find the boundary of each subdomain, the user should select a pixel inside that subdomain in the input image. By this, the BA finds the pixels located on the boundary of the subdomain using the same method explained earlier. This process continues as long as the user selects a pixel in a new subdomain.

### 3.2.3 Detection of Discontinuities in a Given Domain

The function BA can detect any discontinuity, such as holes, cracks, etc., in a given domain. For this purpose, if any discontinuity exists, the user should select a pixel in each discontinuity in the input image. After this, using the same method as described before, BA finds the boundary of each selected discontinuity. By this, the application detects discontinuities and excludes them from the main domain. To this end, after meshing the structure, the application finds all elements inside the discontinuity region and excludes them from the model.

### 3.2.4 Development of a Corrugated Model

In a recent study, we developed a method for modeling out-of-plane (i.e., corrugated) domains [58]. Here, we modified this technique to make it more efficient and easier to implement. This technique requires an additional input image with the same frame size as the main input image. The other image should include information regarding the corrugation of the out-of-plane domain. The information should include the location of the maximum and minimum heights, indicated by the black and white colors, respectively. The value of pixels in the secondary image, therefore, serves as a measure of the height of that pixel: pixel values 0 and 1 indicate the maximum and minimum heights, respectively.

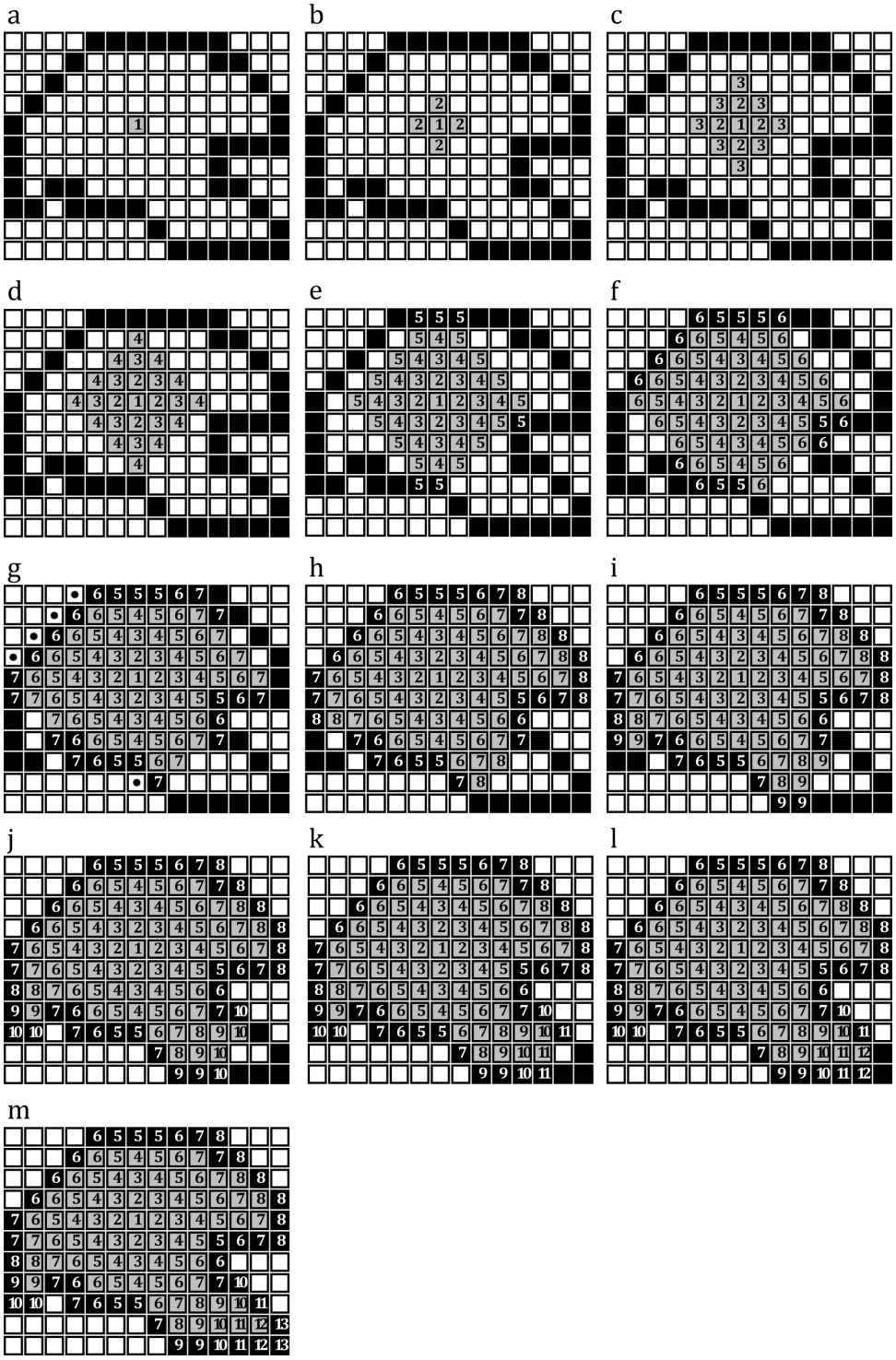



Figure 3.1: Detection of the border of an arbitrary domain using the Burning Algorithm. (a) A white pixel inside the domain is selected. (b) The BA searches for white and black pixels around the selected pixel in four orthogonal directions. (c) The BA searches for white and black pixels around the detected white pixels in the previous iteration. (d–m) This process continues until there is no white pixel inside the domain (m).

If there is more than one maximum or minimum height in a domain, any local extremum can be marked in grey color. The intensity of the grey color in each local extremum indicates the relative height of that extremum compared to the absolute extremum. The recursive Equation 3.1 is used to smooth the corrugations to avoid any abrupt change in the height of a model at the location of an extremum.

$$v(r,c) = mean \sum_{i=-1}^{1} \sum_{j=-1}^{1} v(r+i, c+j) \tag{3.1}$$

where $r$ and $c$ represent the number of the row and column of a pixel in the image, respectively. $v$ is the value of that pixel. $i$ and $j$ are the index of the row and column of the pixels in the image. This equation recursively updates the color intensity of the pixels in the secondary image and, thereby, the height of those pixels in the model. The number of iterations, which is set by the user, controls the sharpness of corrugations in the developed model. The values of pixels in the secondary input image, which are between 0 and 1, represent the relative heights of corrugations. Figure 3.2a shows a corrugated object. The image of the object from the top view is shown in Figure 3.2b. Figure 3.2c shows the secondary input image, which has the same frame size as the image shown in Figure 3.2b. The black line in the middle of the image represents the position of the only available height maximum, and the white color corresponds to the regions with the minimum height. When using these two images, the application develops a model similar to that shown in Figure 3.2d. Figure 3.2j shows the gradual changes in the corrugation of the model by the use of Equation 3.1 after 20, 100, 150, 200, and 300 iterations (Figure 3.2e–i).

### 3.2.5 Mesh Generation

distmesh2d is a mesh generator in MATLAB that employs a distance function, d(x,y), to describe the geometry of a domain [99]. The Delaunay algorithm is used in distmesh2d to generate triangular meshes. The first line of the code distmesh2d, the calling syntax, represents the inputs and outputs of the MATLAB code:

*function [p,t]= distmesh2d(fd,fh,h0,bbox,pfix)*

where the input arguments are as follows:

I. fd, the distance function that defines the boundary of the domain.

II. fh, the distance function, which controls the convergence of the size of elements. The size of the elements decreases near fh.

III. h0, the distance between nodes in the initial distribution.

IV. bbox, the bounding box in which the domain is located.

V. pfix, defines nodal points, which are set as fixed points while generating elements.

distmesh2d produces the following outputs:

I. p, gives the coordinate of the nodal points.

II. t, indicates the connection between the nodes.

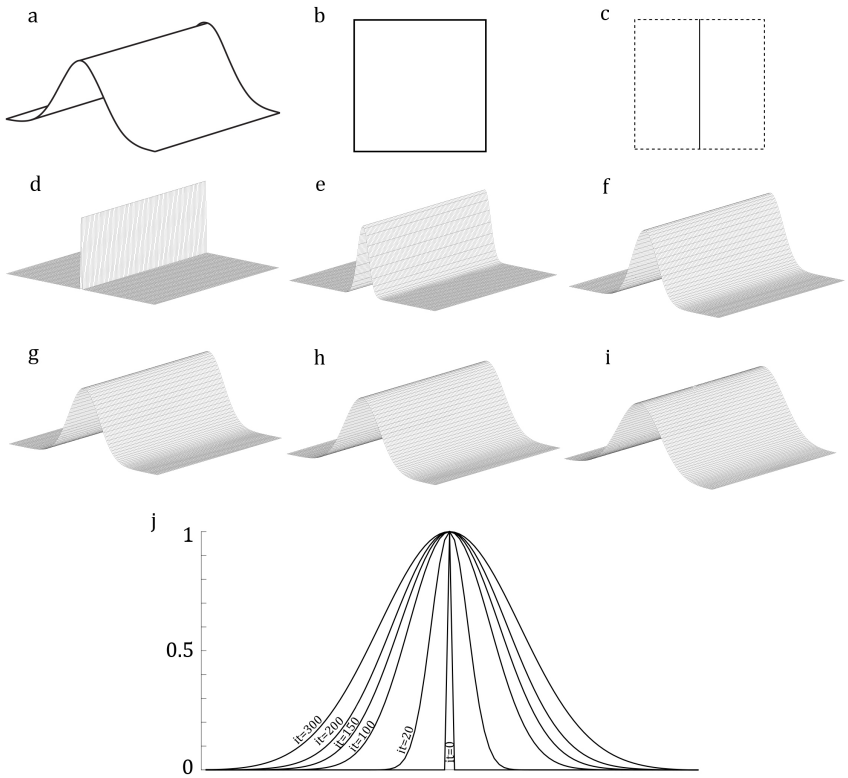



Figure 3.2: Modeling of an out-of-plane domain. (a) An out-of-plane domain. (b) A top-view image of the domain. The image is used by the BA to detect the domain. (c) The secondary image contains a black line that represents the maximum height. The regions with zero height are colored in white. (d) A developed model based on the input images. (e–i) Smoothing the height of the meshed model using the iterative algorithm. (j) Changes in the corrugation pattern in different iterations.

Here, coordinates of the nodes on the boundary of a given domain, which are obtained by the BA, are used to define the distance functions fd and fh for the mesh generator distmesh2d. In addition to the distance functions fd and fh, distmesh2d has three other inputs: h0, bbox, and pfix. h0, the distance between initial nodes, can be set to 1, because the minimum distance between two pixels is 1. bbox is equal to the frame size of the imported image (size of the input matrix). The pixels located on the boundaries of the subdomains, extracted by the BA, are defined as fixed points, pfix. distmesh2d can generate both structured and unstructured elements. However, in this study, we set it to create only unstructured elements, because this type of element fits better with our aim for developing models of geometrically complex structures.

### 3.2.6 Outputs

WingMesh generates an INP file (i.e., an Abaqus input file), which contains information regarding the coordinates of the nodal points, their connections, type of elements, sections of the domain, and the material properties of sections. Detailed information about INP files can be found in the Supplementary Data [100].

## 3.3 Graphical User Interface

WingMesh was coded in MATLAB 2019a, and MATLAB App Designer was employed to develop a GUI. This GUI makes the method easy to implement and eliminates the need to know a programming language. The EXE file of WingMesh, and its description are available as *WingMesh-Method S1* in the Supplementary Data [100]. Besides *WingMesh Video S2* comprehensively describes WingMesh and its GUI.

## 3.4 Examples

- **Example 1.** An in-plane domain Figure 3.3a shows a single in-plane domain with straight-line borders and sharp corners. Figure 3.3b shows the output model. The INP output file developed by the method is available in *WingMesh-Model S1*.
- **Example 2.** An in-plane domain consisting of two subdomains Figure 3.3c illustrates the same domain shown in Figure 3.3a, which is subdivided into two subdomains. As shown in Figure 3.3d, WingMesh was able to detect the border between the subdomains. The subdomains have meshed separately as two sections of a single model. The generated INP file is available in *WingMesh-Model S2*.
- **Example 3.** An in-plane domain with subdomains and a discontinuity We added a circular hole within one of the two subdomains of the domain given in the previous example (Figure 3.3e). After meshing all subdomains, including the discontinuity in the main domain, the elements generated in the discontinuity were removed before the final model was developed (Figure 3.3f). The INP file is available in *WingMesh-Model S3*.
- **Example 4.** An irregular-shaped in-plane domain with several discontinuities Figure 3.3g illustrates an irregular-shaped domain with curved borders, which contains four discontinuities. Previously, it was impossible to model such an irregular domain with complex-shaped discontinuities using the mesh generator distmesh2d. However, the use of the DIP technique enables WingMesh to mesh such geometries. Figure 3.3h shows the meshed model developed based on the given domain. The INP file is available in *WingMesh-Model S4*.
- **Example 5.** A complex-shaped in-plane domain with several subdomains Figure 3.3i shows the world map with irregularly shaped continents. The meshed model, which is apparently in good agreement with the given image, is presented in Figure 3.3j. The INP file is available in *WingMesh-Model S5*.
- **Example 6.** An asymmetric out-of-plane domain with one height maximum and one height minimum. In this example and the next three cases, we used the domain shown in Figure 3.3a to develop out-of-plane models with different corrugation patterns. Here, we used the image in Figure 3.4a as the secondary input image to provide information on the corrugation spots. The grey color in this image indicates regions with zero height. The black and white lines indicate a height maximum and a height minimum, respectively. Figure 3.4b,c shows the perspective and side views of the meshed model. The INP file is available in *WingMesh-Model S6*.

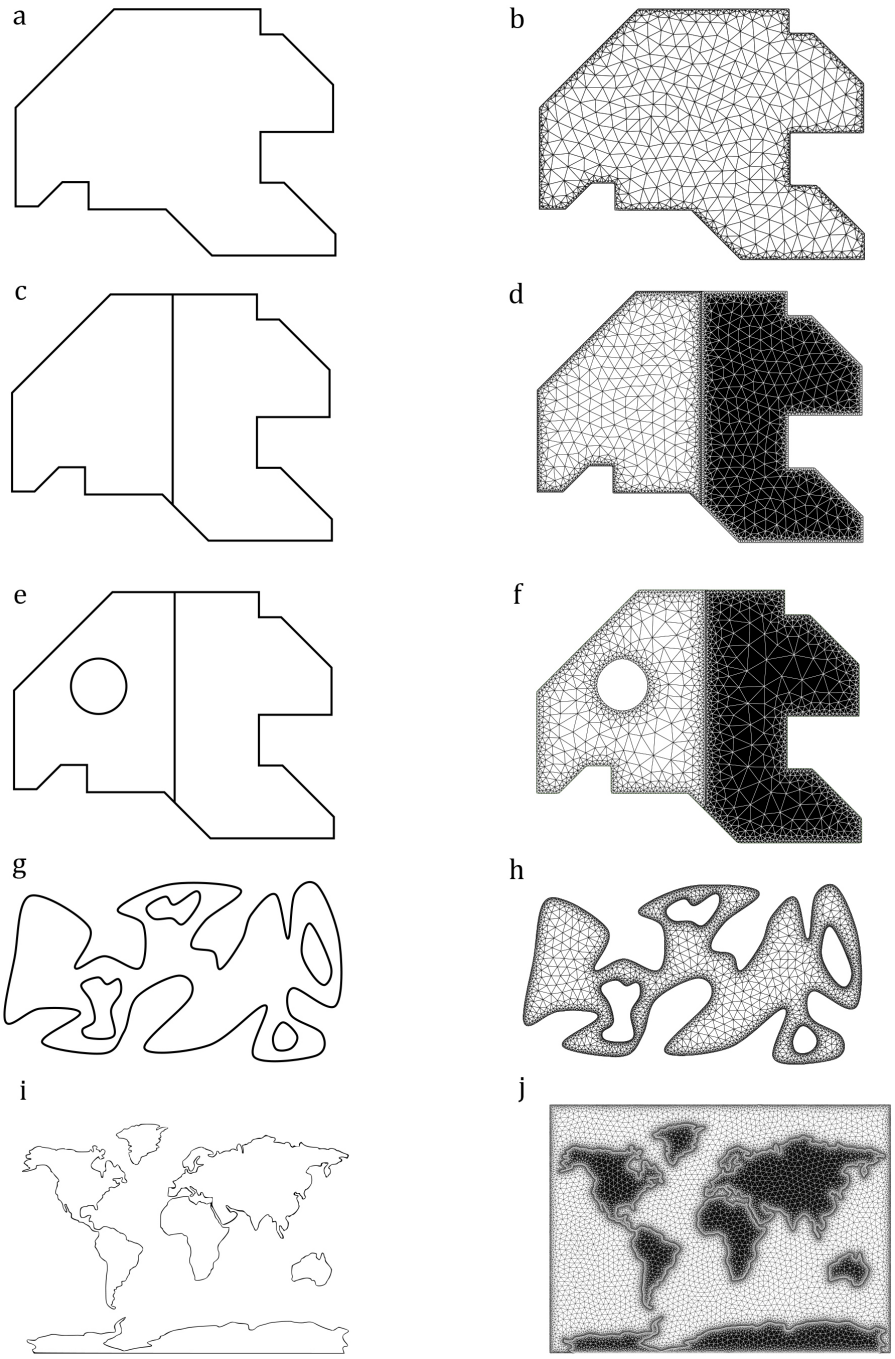



Figure 3.3: Modeling of in-plane domains. (a) Image of a simple in-plane domain. (b) The meshed model developed from the image in (a). (c) Image of an in-plane domain consisting of two subdomains. (d) The meshed model developed from the image in (c). (e) Image of an in-plane domain with two subdomains and a discontinuity. (f) The meshed model developed from the image in (e). (g) Image of an irregular-shaped domain with several discontinuities. (h) The meshed model developed from the image in (g). (i) Image of a complex-shaped in-plane domain with several subdomains. (j) The meshed model developed from the image in (i).

- **Example 7.** An out-of-plane domain with two height maxima. Figure 3.4d shows an image with the black and dark grey lines, which represent two height extrema. Using this as a secondary image results in the development of the meshed model shown in Figure 3.4e,f. The INP file is available in *WingMesh-Model S7*.

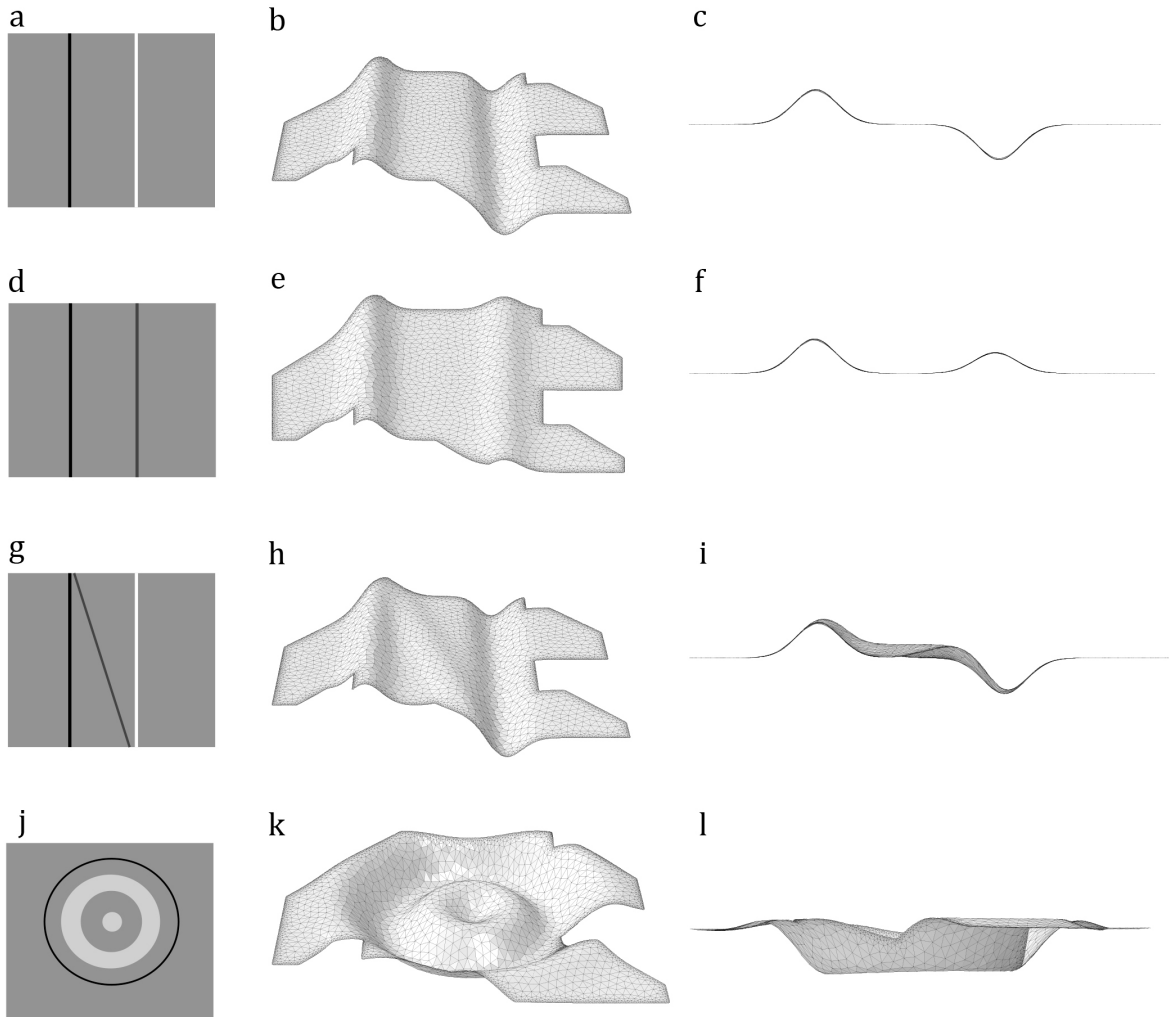



Figure 3.4: Modeling of out-of-plane domains. The use of different secondary images in combination with the same input image, as shown in Figure 3.3a, results in the development of models with different corrugated patterns. (a,d,g,j) Secondary images contain information on corrugation spots. (b,e,h,k) Perspective views of the meshed models created based on the image shown in Figure 3.3a and secondary images shown in Figure 3.4a,d,g,j. (c,f,i,l) Side views of meshed models.

- **Example 8.** An out-of-plane domain with two height maxima and a height minimum. In Figure 3.4g, we added a tilted grey line to those in the secondary image shown in Figure 3.4a. The grey line is expected to change the corrugation pattern of the meshed model in Figure 3.4b by adding a region with a height maximum. Figure 3.4h,i shows the perspective and side views of the model developed using the secondary image in Figure 3.4g. The INP file is available in *WingMesh-Model S8*.

- **Example 9.** An out-of-plane domain with circumferentially oriented height extrema. In this example, we aimed to test the precision of our method by developing a more complex corrugated domain. In this domain, the corrugation spot is circumferentially oriented compared with the other domains that had longitudinal corrugations. Figure 3.4k,l shows the model developed by using Figure 3.4j. The INP file is available in *WingMesh-Model S9*.

- **Example 10.** A beetle wing. Figure 3.5a shows the hindwing of a beetle, *Al-*

*lomyrina dichotoma* (Coleoptera: Scarabaeidae). Figure 3.5b shows the secondary image that was used for generating the corrugation on the model. The black lines show the location of elevated longitudinal veins in comparison with the membranes. The use of Figure 3.5b as a secondary input image results in the development of a model that is shown in Figure 3.5c,d from both dorsal and ventral sides.

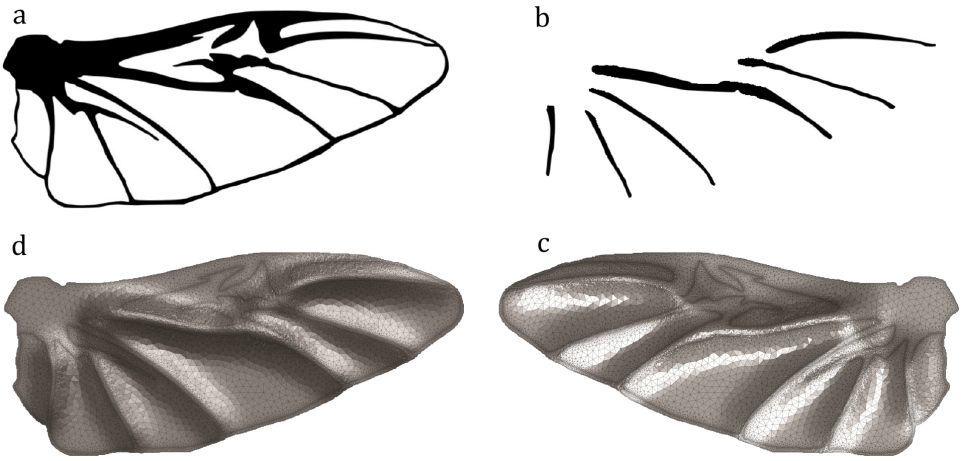



Figure 3.5: Modeling of the hindwing of the beetle *Allomyrina dichotoma* (Coleoptera: Scarabaeidae). (a) Black and white image of the wing. (b) The secondary image for generating corrugations showing the location of the elevated veins. (c) Dorsal view of the generated model. (d) Ventral view of the generated model.

## 3.5 Advantages of WingMesh

WingMesh offers several advantages over existing manual modeling techniques using commercial software packages, such as CATIA, SolidWorks, Abaqus, etc.

- The application is user-friendly and can remarkably reduce the modeling costs.
- Two-dimensional modeling using WingMesh is possible by the use of only an image of a given domain.
- Modeling three-dimensional (3D) out-of-plane domains is simple and can be done by the use of one additional image that contains information on corrugated spots.
- WingMesh can develop meshed models of domains that consist of several subdomains and discontinuities.
- WingMesh is particularly useful for modeling a large number of insect wings for comparative investigations.
- Considering the use of computer vision to extract geometric wing features, WingMesh is applicable for insect wings that contain a high degree of geometric complexity.
- The input image for WingMesh should have only sufficient resolution. This is in contrast to existing tools for extracting morphological features of insect wings using an image, which usually requires high-resolution images at a large size [6, 56, 58].

WingMesh has improved the applicability of distmesh2d, as listed below:

- Extracting the distance function for complex geometries is a time-consuming and error-prone task, which has been overcome by the use of the computer vision in WingMesh.
- WingMesh generates a INP file as the output, which is a frequently used file format.
- WingMesh has an improved ability to mesh structures that contain many discontinuities. This ability was poor in distmesh2d, especially when dealing with domains with more than one discontinuity.
- In contrast to distmesh2d, that can mesh domains that have no subdomains, WingMesh is capable of modeling domains with numerous subdomains.
- Compared with distmesh2d, WingMesh can model out-of-plane domains.

## 3.6 Applications of WingMesh

The application presented in this study can be used for modeling a wide range of objects in both science and engineering, where a planar FE model is required. For example, models developed by our application could be used to understand the mechanical behavior of biological structures, such as insect wings, plant leaves, etc. It can be employed in engineering for FE modeling of plate and shell structures used in aircraft, space crafts, ships, pressure vessels, etc. (see [101, 102] for more examples). Our method could also be used in geology and geo-mechanics for the prediction of the mechanical response of complex heterogeneous rock and concrete structures. Although WingMesh is a promising first step towards the automatic modeling of insect wings, there still remain some other structural features that can be included in a wing model. A few examples of such features are nodus and vein joints, which play key roles in wing deformations both during flight [1, 103, 104] and at rest (i.e., wing folding [9, 105, 106]). Hence, as developers of WingMesh, we are currently working to develop the next generation of our program, which is able to create wing models with more structural details.

# Chapter 4

# *Paper II.* An Image Based Application in MATLAB for Automated Modeling and Morphological Analysis of Insect Wings[1]


## Summary

Despite extensive research on the biomechanics of insect wings over the past years, direct mechanical measurements on sensitive wing specimens remain very challenging. This is especially true for examining delicate museum specimens. This has made the finite element method popular in studies of wing biomechanics. Considering the complexities of insect wings, developing a wing model is usually error-prone and time-consuming. Hence, numerical studies in this area have often accompanied oversimplified models. Here we address this challenge by developing a new tool for fast, precise modeling of insect wings. This application, called WingGram, uses computer vision to detect the boundaries of wings and wing cells from a 2D image. The app can be used to develop wing models that include complex venations, corrugations and camber. WingGram can extract geometric features of the wings, including dimensions of the wing domain and subdomains and the location of vein junctions. Allowing researchers to simply model wings with a variety of forms, shapes and sizes, our application can facilitate studies of insect wing morphology and biomechanics. Being an open-access resource, WingGram has a unique application to expand how scientists, educators, and industry professionals analyse insect wings and similar shell structures in other fields, such as aerospace.



[1]This chapter has been published in *Scientific Reports (Nature)* under an open access license on 17 August 2022. Reproduced with permission from Springer Nature.
Citation: S Eshghi, F Nabati, S Shafaghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. An image based application in MATLAB for automated modelling and morphological analysis of insect wings. *Scientific Reports*, 12(1):13917, 2022. https://doi.org/10.1038/s41598-022-17859-9.

## 4.1 Introduction

Insect wings have various shapes and venation patterns [107, 108]. Many studies have aimed to quantify the link between the morphology of the wings and flight performance of insects, including the migratory behavior [5, 108, 109], aerodynamic force generation [110–114], territory defense and resource holding [115], mating behavior [116], maneuverability and agility [113, 117] and damage tolerance [112, 118]. An essential step in this kind of research is to quantify the shape of the wings, their subdomains (known as cells), and the location of the junctions (also known as joints, vein joints or micro joints). However, this is a challenging and time-consuming task, especially for the geometrically complex wings, such as the wings of dragonflies, mayflies, and locusts, which have many wing cells. In recent years, a few methods have been developed based on deep learning and computer vision to extract the geometric properties of insect wings. DrawWing, for example, is an app that extracts the location of junctions of bee wings [56]. A simple developmental model has also been presented recently to extract the venation pattern of insect wings [43, 55]. FijiWings is another explicitly designed example for studying *Drosophila* wings [57]. NET and NEFI are network extraction apps that can extract the location of the junctions [65, 66]. All these apps can be used to perform a specific task or for a particular wing type. Insect wings have also attracted much attention in the field of biomechanics. Considering the technical challenges associated with direct mechanical measurements on insect wings, many studies in this area have used finite element modeling [2, 5, 13, 34, 39, 42, 95, 108, 119]. Finite element software packages, such as SolidWorks, Abaqus and Catia, enable users to develop insect wings models manually. Considering that insect wings are not 2D planar structures but are somewhat wrinkled by intricate patterns of corrugations, which drastically influence their function [1, 41, 120, 121], manual modeling of wings is a time-consuming and error-prone process, even for an expert user. Hence, many previous efforts have resulted in the development of wing models with huge oversimplifications [2, 6, 34, 35, 39, 41, 119, 122]. To the best of our knowledge, the only existing semi-automated tool for finite element modeling insect wings is our previous app, WingMesh [59]. However, WingMesh represents a significant drawback: the absence of control over the type and size of the produced mesh. To overcome this methodological gap, here we present a powerful app, WingGram, for extracting the geometric properties of insect wings and developing precise finite element wing models. We use examples of ten insect wings with different levels of geometric complexity to evaluate the applicability of WingGram.

## 4.2 Methods and Concepts

WingGram is an open-access app with a user-friendly graphical user interface (GUI) that we created in App Designer in MATLAB 2020b. The fast marching algorithm [71, 72], a well-known method in computer vision, is used to extract the outer edges of an insect wing (here referred to as domain) and those of wing cells (here referred to as subdomains) using only an input 2D image. The Ramer-Douglas-Peucker line simplification algorithm is employed to reduce the saturation of points on the detected boundaries for increasing the runtime [83, 84]. We improved our previous method, WingMesh [58, 59], and included it in WingGram to add higher flexibility in modeling 3D corrugations. Python scripting in Abaqus is used to generate finite element models. The fast marching algorithm extracts the wing cell's geometric properties. The Zhang-Suen thinning algorithm [123] is used to skeletonize the lines in an input wing image and detect the location of vein junctions.

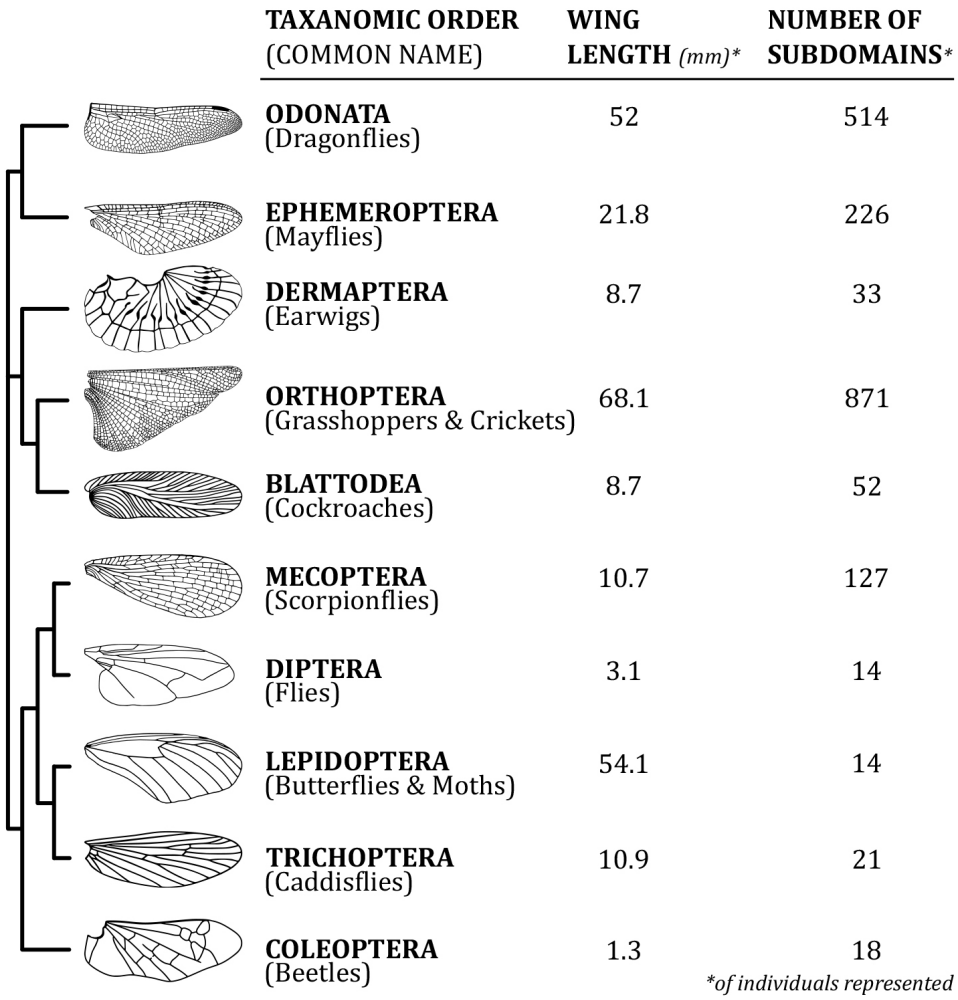



Figure 4.1: Wings from ten representative insect orders were analyzed and modeled using WingGram (wing drawings were taken from [43]).

The finite element simulation from Combes and Daniel (2003) is used to validate the generated model by WingGram [5].

### 4.2.1 Extracting the Boundary of the Wing and Wing Cells

The fast marching algorithm is a recursive method in computer vision and detects an arbitrary domain [71, 72]. In WingGram, we use this algorithm to detect the boundary of insect wings, their wing cells, and any discontinuity within them. Figure 4.2a illustrates how the fast marching algorithm works. Each square and number inside each figure represents a pixel and the iteration phases, respectively. Pixel 1 is an arbitrary white pixel inside the domain. When the first white pixel is selected, the code searches for white pixels around that pixel in the four cardinal directions (i.e. right, left, up and down)(Figure 4.2b). When the code finds white pixels around pixel 1, it searches for the neighbour black pixels. However, this time, it checks all eight pixels around pixel 1 (i.e. Figure 4.2c, pixels 2 to 9). This strategy distinguishes the neighbour cells, which might not be possible otherwise (Figure 4.2d,e). The code stores the coordinates of the found white pixels and uses them as initial pixels for the next iteration. The app also keeps the coordinates of the located black pixels as the domain's boundary. In each iteration, the algorithm changes the colors of the found white and black pixels to light grey and dark

grey, respectively, to avoid duplication. A similar procedure continues until there are no undetected white pixels inside the wing.

### 4.2.2 Ramer-Douglas-Peucker Line Simplification Algorithm

Using the fast marching algorithm, WingGram could detect the pixels located on the boundary of a domain. However, the presence of all these pixels is not required. The higher the number of pixels at the boundary, the higher the computational complexity. Ramer-Douglas-Peucker line simplification is a recursive method that keeps only the critical points on the boundary and removes the rest [83, 84]. This algorithm creates a similar curve, as the original one, with fewer numbers of pixels. In this algorithm, $\epsilon$ is a criterion of similarity that defines the Hausdorff distance, i.e. the maximum length, between the original curve and the simplified one [124]. A smaller $\epsilon$ allows more points to remain on the curve. Figure 4.2f illustrates how this method works on an arbitrary domain. In the first step, this algorithm marks the first and the last points of the curve (the two points overlap in a closed curve). Point P1 in the first iteration is the first found point. Then, the algorithm finds the farthest point from the first kept point (Figure 4.2f, iteration 2, point P2). If the distance between two found points is more than $\epsilon$, the algorithm marks and keeps the new point. The algorithm then subdivides the original curve by connecting the two marked points. In each part, the algorithm finds the farthest point from the line between P1 and P2 and marks them as new fixed points if their distance from the line between P1 and P2 is more than $\epsilon$ (Figure 4.2f, iteration 3). The algorithm recursively continues the same procedure for new segments until there are no points in which the distance of the point from the first and the last points of that segment is more than $\epsilon$ (Figure 4.2f, iterations 4 to 8). In the last iteration, only 17 out of 34 initial points remained without any noticeable change in the curve. Code S13 is the MATLAB code developed for line simplification method.

### 4.2.3 Corrugation Assignment

WingGram generates 3D finite element models of an insect wing by modeling wing corrugations. For this purpose, the app needs a secondary image. To make the secondary image, the user can copy the main image, highlight the place of corrugations, and then erase the rest of the image (Figure 4.2g-i). The app subdivides the secondary image into several vertical sections (Figure 4.2i). In the secondary image, the colour of each pixel represents the height of that pixel (Z coordinate), with white being in the valleys and black in the hills. Like in [58, 59], a smoothing method is applied to avoid sudden height changes by introducing a greyish fade margin next to the height maxima. Each section represents a curve(Figure 4.2j). When the curves connect, they form a continuous corrugated plate. Then, the boundary of the corrugated plate is defined according to that of the main domain in the original input image, which results in a 3D corrugated model (Figure 4.2k). See *WingGram Video S1* for a visual introduction to preparing the secondary image [100].

### 4.2.4 Generating Models Using Python Scripting in Abaqus

The journal file (i.e., JNL file) of Abaqus is a Python script containing all information regarding a generated model's geometry. WingGram generates 2D-planar and 3D-shell finite element models by adapting the corresponding commands of Abaqus Python scripts. The user can define two sections for all veins and all membranes or multiple sections.

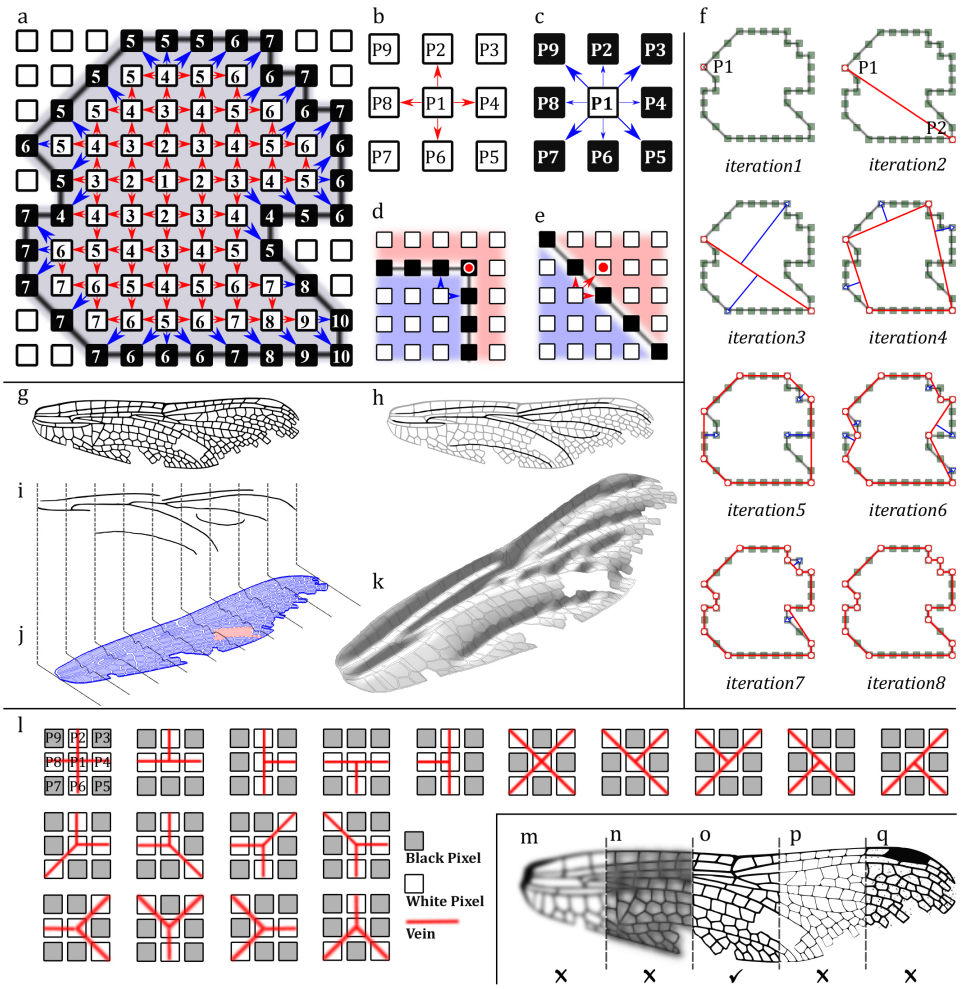



Figure 4.2: Techniques used in WingGram. (a) The fast marching algorithm for detecting the boundary of a domain. (b) cardinal directions to find new white pixels in the fast marching algorithm. (c) All directions to find new black pixels in the fast marching algorithm. (d) The fast marching algorithm may miss the point if only the cardinal directions check for finding new black pixels. (e) Percolating to neighbor domains may occur if the fast marching algorithm checks all the directions for finding new white pixels. (f) Ramer-Douglas-Peucker line simplification algorithm. (g-k) Assignment of wing corrugations. (g) The wing image. (h) Highlighted corrugated spots. (i) The secondary image (divided into several sections). (j) Assignment of sections to the main image. (k) The developed corrugated model. (l) Defined conditions for the identification of vein junctions. (m-q) Suitable and non-suitable images. (m) fade image. (n) Dark image. (o) Appropriate image. (p) Thin and unclear venations. (q) Salt and pepper noise.

All required commands to generate any of the two types of models are extracted and embedded in WingGram. The app automatically updates the Python scripts to develop a wing model based on the information extracted from an input image. The app uses coordinates extracted from the fast marching algorithm to define veins, membranes, and discontinuities. Python commands regarding the SHELL LOFT tool are used to assign corrugations. For this purpose, at least two loft sections and one loft path are required. The application uses extracted curved sections from the secondary image as loft sections, and a linear path is defined as the loft path. The app uses the boundaries extracted from the image to determine the main domain (wing outer boundary), subdomain (wing cells), and discontinuities of the final model. After importing the model into Abaqus, the user can assign material properties to veins and membranes, set boundary conditions and loading, generate a required mesh, and start simulations.

### 4.2.5 Measurement of Geometric Properties

After identifying the wing's geometry, WingGram uses the obtained data to quantify wing geometric parameters, including wing cell area, length, and width. The app measures the maximum distance between two points on the boundary of a wing cell as the length. The ratio of the cell area to the cell length is the width of the cell. The cell area is determined using the method described by Bourke for irregular polygons [125]. In this method, WingGram considers a closed polygon made up of lines between N vertices (x_i.y_i ); i is an integer between 0 and N-1.

$$\sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1} y_i) \tag{4.1}$$

### 4.2.6 Identifying Vein Junctions

To identify the location of the vein junction, i.e. where wing veins intersect, the Zhang-Suen thinning algorithm is used to skeletonise the vein [81] (Code S22). After the skeletonisation, the width of all lines shrinks to one pixel. The app considers a pixel in the skeletonized image as a vein junction if it meets any conditions illustrated in Figure 4.2l. For instance, in the first condition, if P1, P2, P4, P6, and P8 are white and P3, P5, P7, and P9 are black, P1 is a vein junction.

### 4.2.7 Validation

To validate the methodology, we developed models of the forewing of the moth *Manduca sexta* and used our model to simulate the mechanical response of the wing to loading. Following the combined experimental and numerical study of Combes and Daniel [5]. we assigned homogeneous Young's moduli of $1.5{*}10^{8} Nm^{-2}$ and $2.1{*}10^{12} Nm^{-2}$ to membranes and veins, respectively. The wing was fixed at its base, and a point force of F = 0.003 N was applied to the wing tip. We measured the maximum displacement of the wing and compared that with the experimental and numerical results of the earlier study.

## 4.3 Description of the User Interface

The user interface of WingGram consists of a tab bar and a display panel. The tab-bar has four tabs, including "Home" (Figure 4.3a), "Assign Corrugation" (Figure 4.3b), "Morphology" (Figure 4.3c), and "Generate Model and Figure" (Figure 4.3d). Below the tab bar, a display panel is embedded to show the ongoing processes (see Supplementary Data for the installable executive file of WingGram) [100]. The user instructions and visual descriptions of WingGram are available in the Supplementary Data as *WingGram Video S1* [100].

### 4.3.1 "Home" Tab: Importing an Image and Detecting Discontinuities and Subdomains

The "Home" tab is embedded to import an image of the wing (Figure 4.3a). Here, we imported an image of a dragonfly wing containing discontinuities. WingGram supports non-vectorized TIF, JPG, and PNG image formats. A very high resolution or a binary image is not required. There is no limitation on the size and resolution of the imported

image. The position, orientation, and angle of the wing in the image do not influence the applicability of the app. Figure 4.2m-q shows the criteria for a suitable input image. Faded and dark images or those with salt-and-pepper noises, as shown in Figure 4.2m,n,q, influence the performance of the fast marching algorithm in identifying the subdomains. WingGram works particularly well if the venation patterns are clear. The application is not ideal for analyzing and modeling wings with dark spots or strong pigments. Figure 4.2p shows thin lines in the input image, which can disturb the performance of the fast marching algorithm for differentiating neighbor subdomains. Figure 4.2o shows an example of a suitable image. After importing the image, the image's name and size appear in specific fields below the "Import Image" push button (Figure 4.3a). By importing the image, the code automatically detects the outer boundary of the wing. In the "Discontinuity" panel, the user can mark the place of discontinuities by pushing the "Pointer" button. Activating the "Auto Detection" switch allows automatic detection of subdomains and discontinuities. Mark a domain as the discontinuity turns its color into dark grey. The same procedure is required to detect subdomains using the "Sub-domains" panel. Mark a domain as the subdomain turns its color into light grey. The next step is to reduce the density of points on the boundary of domains and subdomains. The user can see a list of domains in the "Home" Panel. A blue line appears around it in the display panel by clicking on a domain. Turning the knob-pitch on the right side of the list changes the $\epsilon$ regarding the Ramer-Douglas-Peucker line simplification method to increase or decrease the density of points. Red points on the boundary of selected domains show the remaining points after reducing the density. Reducing the number of points on the border reduces the runtime. In this panel, "Arc Accuracy" shows the similarity between the curve of decreasing points and the original curve extracted from the burning algorithm. Also, there are two buttons under the list. The one with a bin icon is for deleting a domain, and the other lets the user apply the same node density for all other domains. The same procedure is available for discontinuities. The user can switch between the list of domains and discontinuities at the top of the list.

### 4.3.2 "Assign Corrugation" Tab: Importing the Secondary Image to Assign Corrugations

Figure 4.3b shows the "Assign corrugation" tab bar. The user imports the secondary image using the "Secondary Image" button. A spinner is embedded to change the number of sections. We embedded two knobs in our app to change the maximum height (i.e. peak) and sharpness of corrugations. To apply changes to single sections, the user can click on a section, turn the "Link Sections" off, and then use the knobs to adjust each maximum height and sharpness parameter.

### 4.3.3 "Morphology" Tab: Extracting the Location of Vein Junctions and Other Geometric Parameters

Figure 4.3c shows the "Morphology" tab for extracting the geometric properties of insect wings. To use this panel, the user has to set the scale (i.e., length per pixel) and the number of histogram bins and then push the "Start Processing" button to extract geometric properties from the image of the wing. As described, the result appears in several figures after the processing ends. The user can switch between the radio buttons to observe each result. Switching between the radio buttons allows visualization of the contours/histograms of the wing cells' area, length, and width. Other radio buttons show

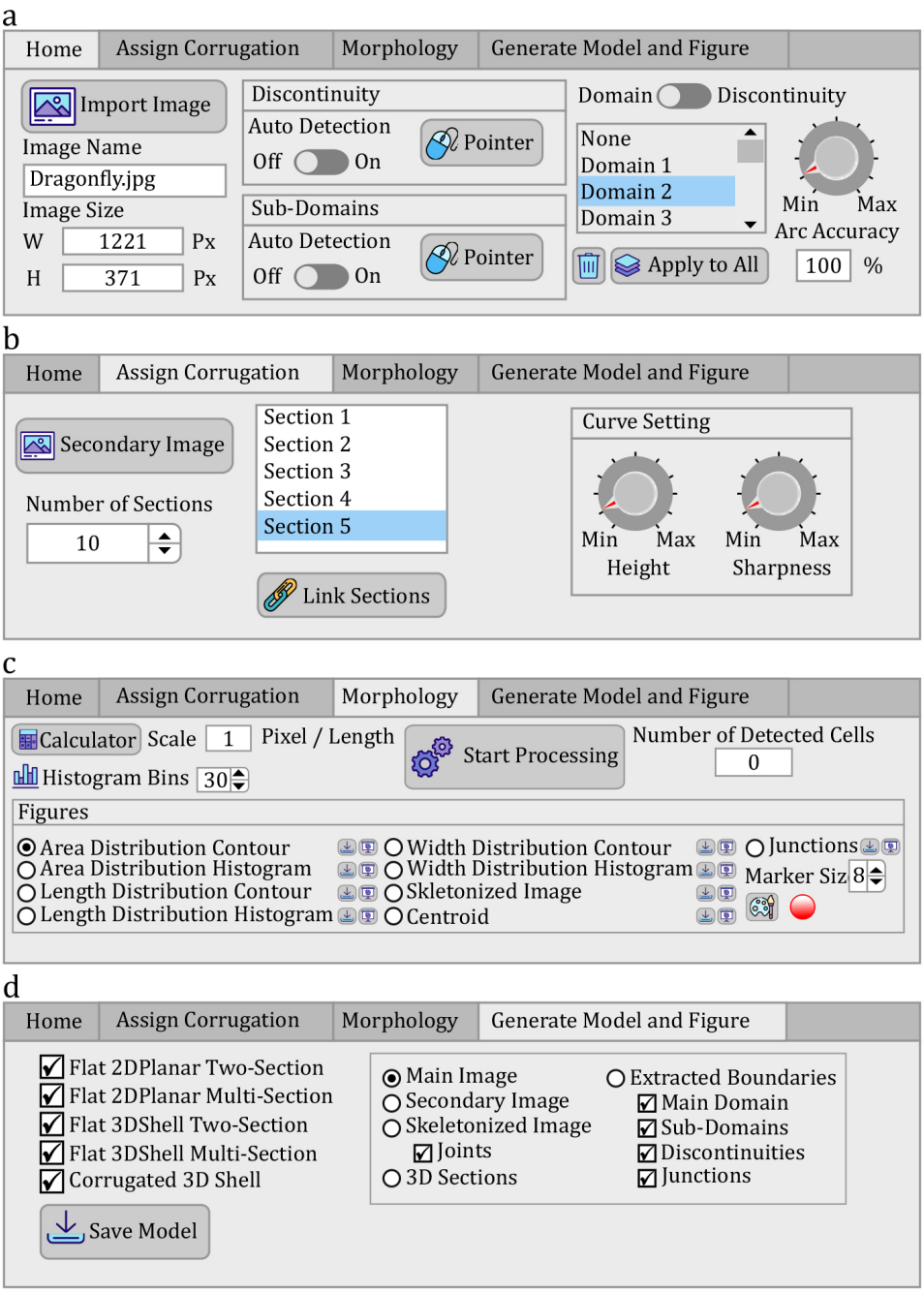



Figure 4.3: The user interface of WingGram. (a) This tab is embedded to import the image, specify the discontinuity and subdomains, and change the number of points on the detected boundaries. (b) The user can import the secondary image to assign corrugations in this tab. Two knobs are embedded to change the height and the sharpness of the corrugation. (c) This tab is embedded to extract the morphological properties of the insect wing. (d) Using this tab, the user can save the FE model of the wing.

the distributions of the vein junctions, the wing centroid's location, and the skeletonized image. There are two buttons on the right side of each radio button. One of them saves the data of the corresponding radio button. It keeps the corresponding matrix for contours, the data of histograms, the coordinates of junctions, and the coordinates of the centroid. The second button opens the corresponding figure in a separate window.

### 4.3.4 "Generate Model and Figures" Tab: Generating a JNL File

We embedded four options for developing a planar model without corrugations, listed below. The last option works if corrugations should be assigned.

- **Two-/multi-section, three-dimensional shell.** This type of model is suitable for applying out-of-plane loadings, such as uniform pressure, impact, out-of-plane point force, and displacement.
- **Two-/multi-section, two-dimensional planar.** This type of model is suitable for applying in-plane loading, such as shear, tensile, or compressive loading.

Figure 4.3d shows a group of available figures. Switching between radio buttons shows figures regarding the selected option. On the "Skeletonised Image" button, the user can hide the presence of junctions. On the "Extracted Boundaries" button, the user can mark the presence of the "Main Domain," "Sub-domains," "Discontinuities," and "Junctions" by turning off corresponding check boxes.

## 4.4 WingGram in Application

Ten representative wing images from different insect orders that had noticeably different shapes and venation patterns were selected to show the performance of WingGram, (Figure 4.1). Wings were scaled to have the same length as 100 pixels. Figure 4.4 illustrates the outcome of the WingGram for each wing, including the wing's FE model, the vein junctions' location, the distributions and histograms of the cell area, cell length, and cell width (see Supplementary Data *WingGram-Model S1* to *WingGram-Model S10* for FE models) [100]. We also anticipate WingGram to apply to studies of damaged wings and fossils. Figures 4.5a and b show a damaged dragonfly wing and the fossil of *Auroradraco eos* [126]. Here, we used WingGram to extract the location of the vein junctions and the area, length, and width of the wing cells for both wings. Also, Using WingGram, we developed a geometric multi-section model of the damaged wing and a two-section model of the damaged wing and the fossil (find FE models in the Supplementary Data *WingGram-Model S11*, *WingGram-Model S12* and *WingGram-Model S13*) [100]. The models are imported into the Abaqus CAE and are meshed by triangular shell elements. In the multi-section model, we assigned different material properties to the cells. The 3D model of the damaged wing using the secondary image is generated (find the FE model in the Supplementary Data *WingGram-Model S14*) [100]. Assigning corrugations to a wing model in WingGram is convenient but might not be very accurate as the user sets them. In Figure 5c, we used a scanning 3D measurement microscope, Keyence VR 3100, to show the 3D pattern on a *Sympetrum* dragonfly hindwing. This image shows the main and secondary image of the wing and the 3D model generated by WingGram (find the FE model in the Supplementary Data *WingGram-Model S15*) [100]. Also, two other secondary images are used for this model to show how the user can add curvature to the whole model, which is available in Figure S3 in the Appendices. WingGram is a modeling app, and virtual models generated by it could be 3D printed to construct physical models. Figure 4.5d shows the main and secondary image of the basal complex of the dragonfly forewing, the virtual model, and the 3D printed model. Due to the limited print area, we isolated the wing model's basal part and removed the thin membranes (See the Supplementary Data *WingGram-Model S16* for the G-Code) [100]. The isolated

part of the wing was fabricated using a Prusa i3 MK3S (Prusa Research s.r.o., Prague, Czech Republic) with white coloured polylactic acid (PLA) filament (Prusa Research s.r.o., Prague, Czech Republic). WingGram is also applicable to many 2D natural and artificial structures, such as leaves, spider nets, geographic maps and/or industrial plates and shells (see Figure S2).

### 4.4.1 Validation Result

Figure S4 shows the result of the FE simulations. The maximum bending displacement of our model is 5.47 mm in the z-direction, which is about 4% different from that obtained by Combes and Daniel [5]. The generated model in Abaqus, with its assigned boundary conditions, material properties, and loading, is available in Supplementary Data as *WingGram-Model S17* [100].

## 4.5 Discussion: Advantages of WingGram Over Other Apps

WingGram equips the user with a combination of tools that can also be found in various other apps. We have also included additional unique options to WingGram, such as semi-automated finite element modeling. Here, we compare WingGram with some of the existing apps.

### 4.5.1 WingGram vs. WingMesh

WingMesh is a MATLAB-based app developed by the authors with a user-friendly interface for finite element modeling of insect wings [59]. WingMesh can generate both 2D-shell and 3D-corrugated shell models of an insect wing. It can include several sections with different material properties and thickness values for the cells in a wing model. Several features make WingGram more powerful than WingMesh. The meshing process in WingMesh cannot be adjusted and often requires a high runtime. Modifying a model developed by WingMesh is not convenient and requires significant effort. WingMesh uses distmesh2d, a well-known meshing tool in MATLAB [99], and computer vision to mesh a wing model. The user doesn't control the type and size of the mesh. The model is not manipulatable in Abaqus because it is made up of orphan meshes. Whilst WingGram uses Python scripting in Abaqus to generate JNL models. After importing the model in Abaqus, the user can use all meshing tools to create an ideal mesh in the model and manipulate the geometry easily. The modeling process in WingMesh is drastically slower than in WingGram because the user must wait for MATLAB to generate a mesh in the geometry that might not be so precise. WingMesh has difficulty in modeling complex wings like dragonfly wings. Because WingGram is more powerful than WingMesh and doesn't abandon any types of insect wings, it has several additional tools that were mentioned earlier in the text. See Figure S5 in Appendices for a comparison between modeling a similar wing with WingGram, and WingMesh. The modeling process from WingMesh took about 6 hours while the same wing needed only 15 minutes to be generated by WingGram.

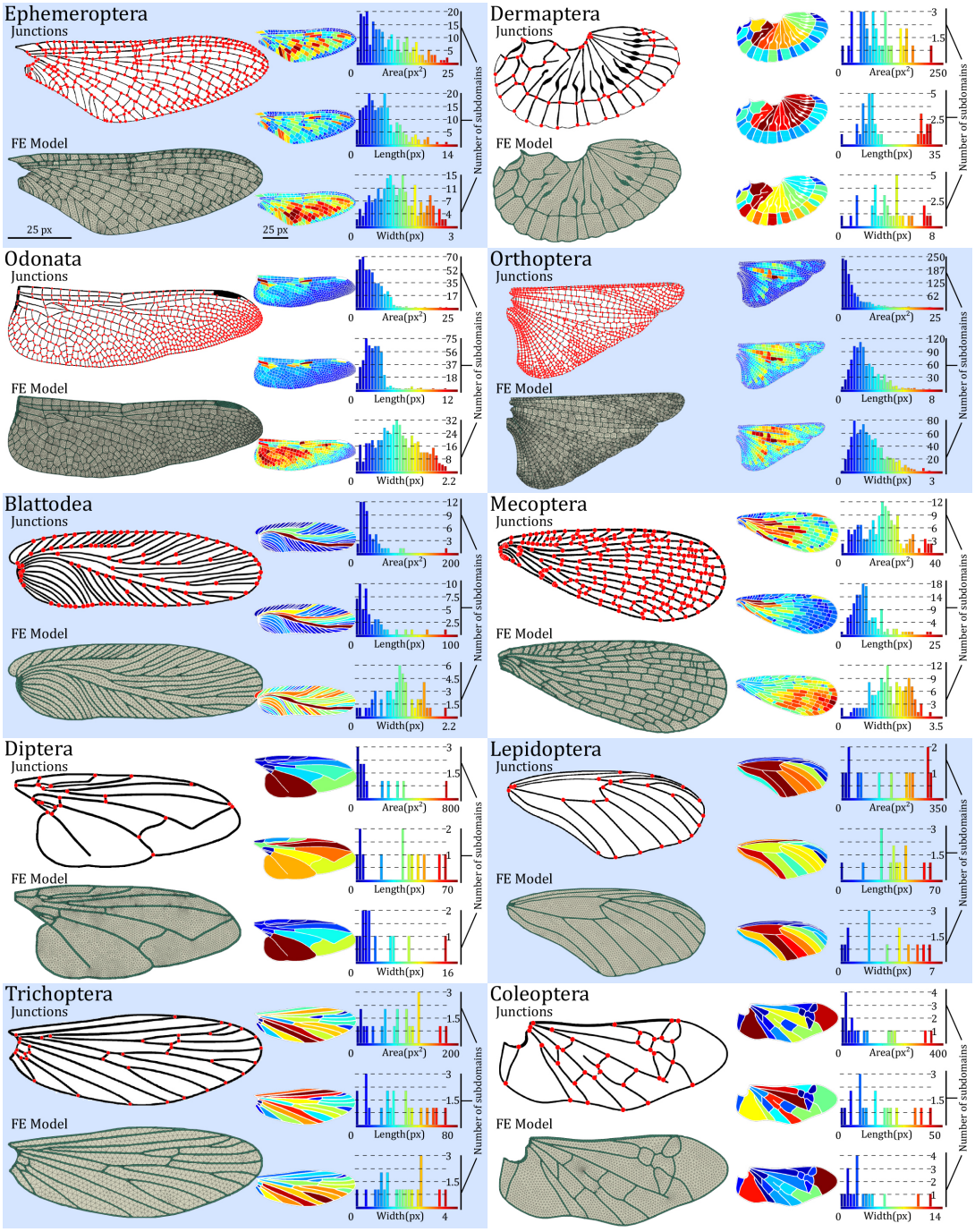



Figure 4.4: The performance of WingGram for extracting the geometric characteristics of ten representative wings from different insect orders. For each wing the FE model, location of vein junctions, area, length, and width distribution contour and histogram are shown. The length of all wings is considered with the same size as 100 pixels. For each wing, three histograms illustrate the distribution of the area ($px^2$)), length ($px$), and width ($px$). Histograms are accompanied by their corresponding contours on the left side. The color bar of each contour is under its corresponding histogram. The place of junctions are extracted for each wing, and the FE model generated by WingGram is meshed in Abaqus. Insect orders are Ephemeroptera, Dermaptera, Odonata, Orthoptera, Blattodea, Mecoptera, Diptera, Lepidoptera, Trichoptera, and Coleoptera.

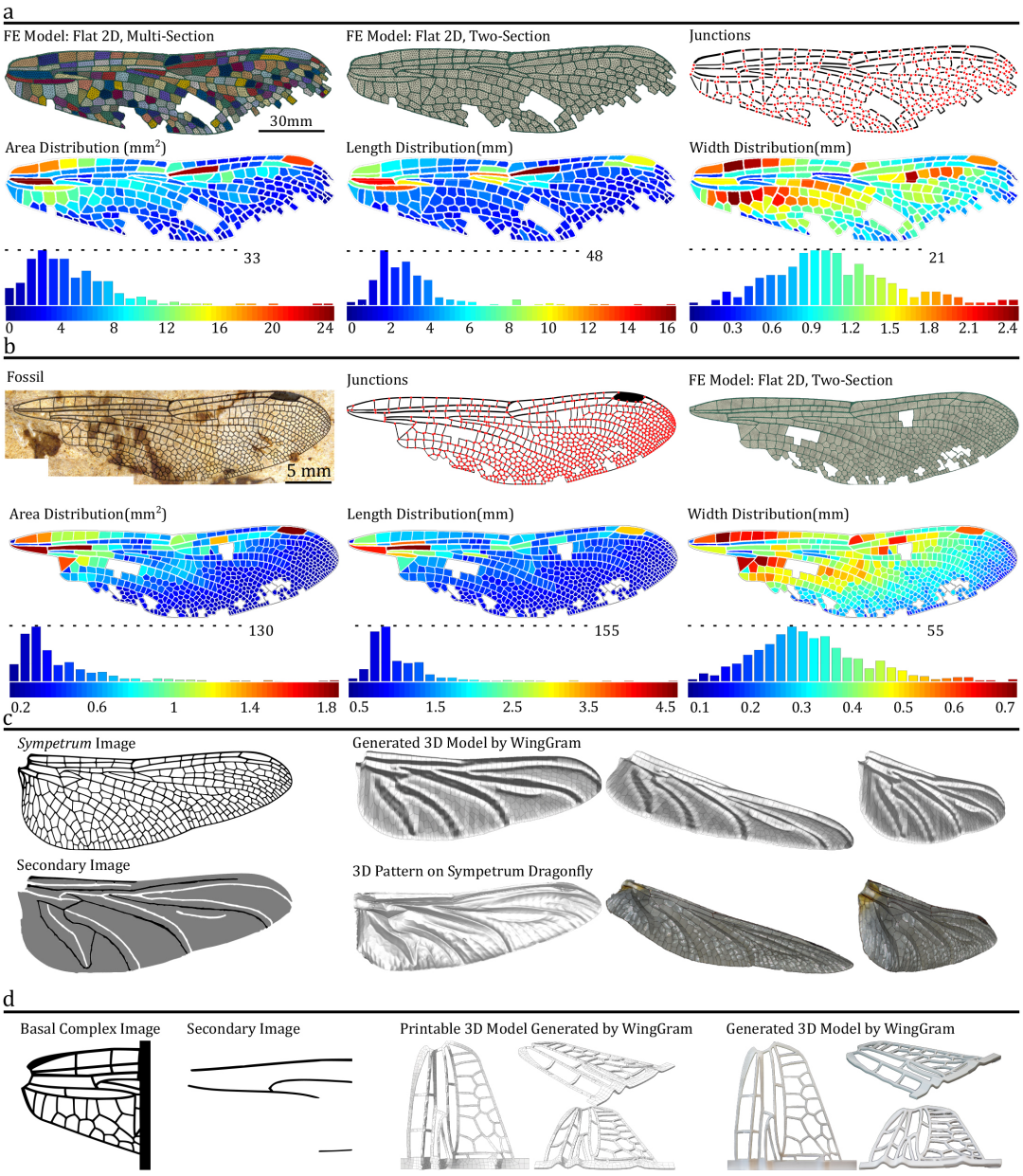



Figure 4.5: The application of WingGram in modeling damaged wings, fossil wings and corrugated wing corrugations, and its use in 3D printing. (a) Damaged wing of dragonfly [73]: main image; secondary image to assign corrugation; Flat 2D, two-section FE model; Flat 2D multi-section FE model; corrugated 3D model; location of vein junctions; area, length, and width distribution contour and histogram. (b) The fossil wing: photograph of *Auroradraco eos* [126]; location of vein junctions; Flat 2D, two-section FE model; area, length, and width distribution contour and histogram. (c) Comparison of the corrugations of a dragonfly wing (*Sympetrum vulgatum*) obtained by a 3D measurement microscope Keyence VR 3100 and WingGram: the image of a *Sympetrum* dragonfly; secondary image for assigning corrugations; the 3D model generated by WingGram; 3D scan by Keyence VR 3100. (d) The 3D printable model generated by WingGram: dragonfly basal complex image; secondary image to assign corrugation; the virtual model of the basal complex; the virtual corrugated model developed by WingGram; 3D printed model of the basal complex.

### 4.5.2 WingGram vs. DrawWing

DrawWing extracts the location of the vein junctions in an insect wing [56]. This app needs a high-resolution image (more than 2400 dpi × 2400 dpi). As mentioned by its developers, DrawWing only works with honeybee wings (*Apis mellifera*). In contrast, WingGram has no limitations regarding image resolution and applies to almost all types of insect wings.

### 4.5.3 WingGram vs. NEFI & NET

These apps extract network data from an image using image processing and computer vision. They are applicable for leaf venation, spider webs, crack paths, insect wing venations, and similar structures. They extract the location of junctions and the connection between them, which is applicable for detecting the location of vein junctions of an insect wing [65, 66]. This is the only common function between the NEFI, NET, and WingGram. Our application works as accurately as NEFI and NET.

### 4.5.4 WingGram vs. FijiWings

FijiWings uses the advantages of the ImageJ Fiji to measure some geometric features of the *Drosophila* wing, like the area of the wing and trichome density. This app needs a high-resolution image to work appropriately [57]. In contrast to FijiWings, WingGram extracts the area, length, and width of subdomains, even if there are damaged input wings, and is applicable for almost all insect wings. WingGram contains unique features applicable to field studies on insect wings regarding the wing's mechanical behavior or morphological properties. To the best of our knowledge, WingGram is the only existing tool that can be used to develop a precise geometric model (not only mesh) using only an image of insect wings. WingGram can facilitate future studies on insect wings and their components, such as ambient-, cross-, and longitudinal veins, corrugations, and membranes. For example, the location of junctions can be used to test fluctuating asymmetry between left and right wings, which is important for ecological studies to monitor environmental pollution [127, 128]. Also, the wing shape and vein junctions' location can be used to identify insect species [56, 129]. Extracting the area, length and width of membranes is another feature of WingGram that can enable us to study the relationship between wing form and design. Although compared with many existing modeling methods, WingGram offers better efficiency and less computational time, it represents some limitations that can still be improved. WingGram models wings with homogenous thickness, while the thickness of different parts of the wing can vary from one place to another. WingGram does not model the cross-section of veins as they are in reality. The authors are currently working on developing a new version of WingGram, which can overcome these limitations and add more features to it.

# Chapter 5

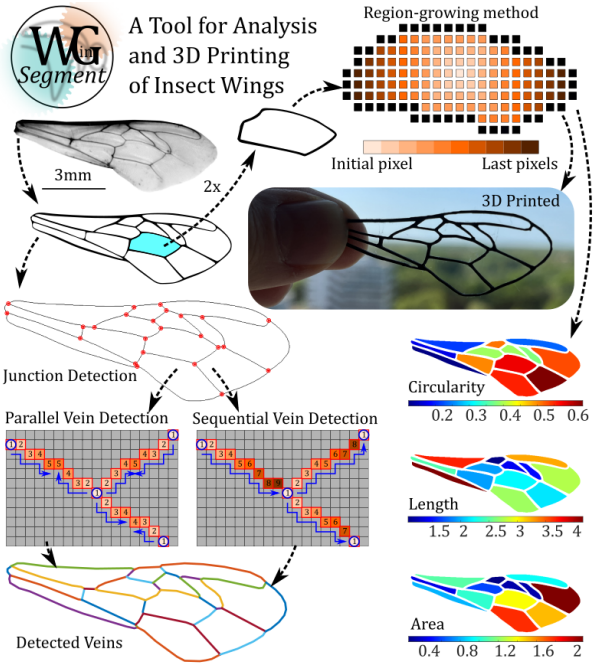
# *Paper III.* WingSegment: A Computer Vision-Based Hybrid Approach for Insect Wing Image Segmentation and 3D Printing[1]


## Summary

This paper introduces WingSegment, a MATLAB App-designed tool employing a hybrid approach of computer vision and graph theory for precise insect wing image segmentation. WingSegment detects cells, junctions, Pterostigma, and venation patterns, measuring geometric features and generating Voronoi patterns. The tool utilizes region growing, thinning, and Dijkstra's algorithms for boundary detection, junction identification, and vein path extraction. It provides histograms and box plots of geometric features, facilitating comprehensive wing analysis. WingSegment's efficiency is validated through comparisons with established tools and manual measurements, demonstrating accurate results. The tool further enables exporting detected boundaries as FreeCAD macro files for 3D modeling and printing, supporting finite element analysis. Beyond advancing insect wing morphology understanding, WingSegment holds broader implications for diverse planar structures, including leaves and geocells. This tool not only enhances automated geometric analysis and 3D model generation in insect wing studies but also contributes to the broader advancement of analysis, 3D printing, and modeling technologies across various planar structures.



[1]This chapter has been published in *Advanced Intelligent Systems (Wiley)* on March 17, 2024. Reproduced with permission from Wiley.
Citation: S Eshghi, H Rajabi, J Poser, and SN Gorb. WingSegment: A Computer Vision-Based Hybrid Approach for Insect Wing Image Segmentation and 3D Printing. *Advanced Intelligent Systems*, 6(5):2300712, 2024. https://doi.org/10.1002/aisy.202300712.




**Short Summary:** WingSegment is a user-friendly software for automated morphometry analysis and 3D modeling of insect wings. It integrates computer vision, image processing, and graph theory to segment insect wing images automatically. Subsequently, it provides statistics related to the size and position of wing cells, junctions, and vein patterns, and generates 3D models in the FCMacro format, compatible with FreeCAD.

## 5.1 Introduction

The study of insect wings presents a captivating challenge, as their intricate structures hold the key to unraveling the secrets of nature's most skilled aviators [1]. Understanding the complex morphology of insect wings is not merely an academic pursuit in multiple scientific domains, such as biomechanics, animal behavior, physiology, and ecology [5, 108–110, 118, 130], but an inspiration for innovative technologies in the field of engineering. These include the development of robust load-bearing structures [131–133], bioinspired joints [25, 95, 134, 135], insect-inspired hinges [26], bioinspired wing [21], insect-inspired composites [136, 137], bioinspired attachment strategies[22–24], and bioinspired grippers [138].

As our understanding of the wing structures deepens, manual analysis methods face challenges in accurately capturing the complex geometric features to characterize wing morphology [139, 140] comprehensively. Furthermore, manual analysis of structure complexity requires precise consideration of geometrical parameters and proves time-consuming and prone to errors. To comprehensively explore the multifaceted nature of insect wings, automated methods powered by computer vision offer a promising avenue for precise and efficient analysis. In recent years, several techniques have emerged that facilitate convenient 3D scanning [141], extraction of geometrical information [43, 55, 60, 73, 74], and automated finite element modeling [58–60] of both insect bodies and wings. These advancements aim to enhance the precision, speed, and efficiency of analyzing insect wing morphology.

NET (Network Extraction Tool) [65], and NEFI (Network Extraction From Image)[66] are tools designed to extract graphs from images of networks found in diverse domains. The kind of graph they extract is a representation of the underlying network structure within the images. They also encompass the aim of extracting networks (e.g. venation pattern) from insect wings. FijiWings [57] is a comprehensive tool that incorporates multiple integrated functions to accurately quantify cell number (trichome count) in the

wings of *Drosophila*. DrawWing [56] is another tool for extracting vein junctions and drawing a fitted diagram of wing outlines and veins. DrawWing is specifically developed to be compatible with the wings of honeybees belonging to the genus *Apis*. However, the aforementioned tools primarily focus on extracting junction locations or venation network patterns, while there are other tools dedicated to automating the modeling of insect wings.

WingMesh [59] is one such tool previously developed by the authors, which aims explicitly at extracting cell boundaries from images of insect wings. It utilizes distmesh2d [99], a mesh generator in MATLAB, to generate a mesh on the insect wing domain. Embedded within WingMesh are methods that facilitate the replication and manipulation of a distinct file format known as INP files, utilized by Abaqus CAE. These methods enable the generation of orphan meshes [142]. Orphan meshes lack the necessary user-friendly functionality to change the wing's geometry easily. It should be noted that generating a wing model using WingMesh is considerably slow. WingGram [60] was later developed as a solution to tackle these challenges, incorporating computer vision techniques to accurately detect wing cell boundaries from two-dimensional images. It introduces a novel approach for assigning corrugations to the wing model using a secondary image. Moreover, WingGram has the capability to extract various geometric features from wings, including cell area, length, width, and vein junction locations. Python scripting in Abaqus was utilized within WingGram to transfer wing geometry from an image to a journal file (JNL). This enables users to import wing geometry without relying on generated meshes, allowing them to leverage the advantages of Abaqus faster and more optimal meshing tools compared to the embedded distmesh2d in WingMesh.

Although the aforementioned tools have been acknowledged for their intended purpose, there is room for improvement further to enhance the analysis and study of insect wings. These enhancements may include extending the detection capabilities beyond venation networks to encompass venation patterns, enhancing robustness and accessibility, and providing additional tools for statistical analysis of insect wings. These improvements would contribute to the advancement of research in this field. Recognizing the limitations of NEFI and NET in detecting curved patterns, along with the species-specific nature of FijiWings [57] and DrawWings [56], and the fact that WingMesh [59] and WingGram [60] exclusively generate Abaqus-compatible models, which may be inaccessible and costly to students, this study aims to address these shortcomings by introducing WingSegment as a fast, accessible, and reliable solution. WingSegment adopts a hybrid approach that integrates computer vision and graph theory to extract cell boundaries, junction coordinates, and vein path coordinates accurately. The methodology encompasses various techniques, such as region growing [71, 72, 79], Dijkstra path finding [143–146], thinning [81], and line simplification [83, 84], which have been employed in this research. Additionally, we have utilized the FreeCAD macro (FC macro) file format [147, 148] to generate automated FC models from insect images, further enhancing the accessibility and flexibility of the software in both modeling and 3D printing of biomimetic wings.

WingSegment additionally produces Voronoi patterns within the wing area by utilizing the centroids of the extracted cells. Voronoi patterns, commonly found in insect wings, hold significant importance in their study [149]. These patterns offer insights into the physical constraints governing the organization of biological tissues, reflecting the crucial role wings play in insect radiation and contributing to their remarkable diversity in shape, size, and color patterns. With their efficient and complete plane tiling, Voronoi patterns serve as convenient models for the intricate structures of insect wings. By examining Voronoi patterns, researchers can gain a valuable understanding of the physical and biological properties of wings, shedding light on insect evolution and adaptation.

## 5.2 Methods

This study proposes a hybrid approach, WingSegment, for segmenting insect wings by integrating various algorithms in image analysis and graph theory. The algorithm is accompanied by a user-friendly graphical interface designed in MATLAB App-designer. WingSegment can detect cells, junctions, and vein paths in the insect wing by taking an image of the wing as input. A region growing algorithm is incorporated into WingSegment to detect the boundary of cells and the wing. The thinning algorithm in image processing is utilized to locate the junctions. A combination of Dijkstra's and thinning algorithms is employed to detect the venation patterns in the wing. In addition, a distance transform function is incorporated to identify the presence of the Pterostigma in the wing. Lastly, a line simplification method compresses the detected cells and venation patterns.

### 5.2.1 Image Preparation

A high-quality image is required since we intend to present this method to segment the insect wing image in detail. A camera, scanner, or microscope can be used for taking images. Figure 5.1a is a sample image of a honeybee taken by a high-resolution scanner (Epson Perfection V850 Pro). Since the method turns the image into a binary image with only black and white pixels with a thresholding of 0.54, the user should ensure that all veins and membranes remain apparent after binarising the image. Figure 5.1b represents an example of an appropriate image for use in WingSegment. Code S2, included in the Appendices, is designed for importing the image and performing the binarization process.

### 5.2.2 Cell Domain Extraction

To segment cell domains, a region growing method is utilized in this study [71, 72, 79, 80]. Figure 5.1d provides an exaggerated pixel view of the domain in Figure 5.1c and illustrates how the method finds a region from a starting point. Five sets are defined to describe the method, including visited white pixels, unvisited white pixels, visited black pixels, unvisited black pixels, and current pixels. The algorithm involves five steps, starting with marking all white pixels as unvisited. Then, a random initial white pixel within a cell is selected as the current pixel, marked as visited, and removed from the unvisited white pixels set. Then, all neighbors' unvisited pixels around the current pixels are checked. If they are white, they are marked as visited white pixels, replaced with the current pixel, and removed from the unvisited white pixels. If they are black pixels, they are marked as visited black pixels and removed from the unvisited black pixels. The algorithm continues while the unvisited white pixels set is not empty, and when it becomes empty, the boundary of that domain is extracted. *WingSegment Video S3* illustrates this procedure on a honeybee wing. In the Appendices, Code S3 depicts the function for implementing the region growing method.

#### Generating Voronoi Pattern

Incorporating a method for extracting the cell boundaries into WingSegment facilitates the calculation of cell centroids. This has been integrated into the WingSegment algorithm, allowing for generating a Voronoi pattern by utilizing the centroids of the segmented cells. By accurately determining the centroids, WingSegment enables the visualization of a Voronoi pattern that represents the spatial distribution of insect wing cells.

### Evaluating Geometric Features

During the process of the boundary extraction, WingSegment simultaneously detects the number of pixels in each boundary. This information is utilized to measure the area of individual cells accurately. Moreover, WingSegment measures the maximum distance inside each boundary, corresponding to the cell's length. Additionally, since the boundary of each cell is already extracted, WingSegment can measure the perimeter of each cell. By utilizing these calculated values and employing equation 1, WingSegment determines the circularity of each cell. The circularity ranges between 0 and 1, where 1 indicates the highest level of circularity, representing how closely a shape resembles a perfect circle [150]. The circularity is quantified using Equation5.1.

$$C = (4\pi * area)/(perimeter^2) \tag{5.1}$$

Here, the *area* represents the measured area of the shape, and the *perimeter* represents the measured perimeter of the shape. In the Appendices, Code S4 represents the function responsible for quantifying all geometric features through the utilization of the region growing method.

### Generating FC Macro File of Detected Boundaries

In FreeCAD, which is a powerful 3D modeling software, the term "FC macro" refers to a macro script written in the FreeCAD scripting language [147, 148]. FreeCAD is an open-source parametric 3D modeling software that allows users to create and modify 3D designs using a graphical interface. However, it also provides a scripting interface allowing users to automate tasks and create custom Python tools. FC macro in FreeCAD is a Python script specifically written to extend the functionality of FreeCAD. Users can create their macros to perform repetitive tasks, automate design processes, or add new features to the software. These macros can be executed within the FreeCAD environment, providing additional functionality beyond what is available through the graphical interface alone.

In addition to automatically detecting cells from the insect wing image, WingSegment incorporates a valuable tool that allows users to export the detected boundaries in an FC macro file format. This feature enables easy transfer of the wing geometry, specifically the boundaries of the detected cells, into FreeCAD. By exporting the boundaries as an FC macro file, users can conveniently import the wing geometry into FreeCAD and perform various operations such as extrusion, generating finite element (FE) models, or creating 3D printable models. This functionality enhances the usability of WingSegment, providing users with a seamless pathway to further analyze and utilize the detected cell boundaries in their design and simulation processes. In the Appendices file, Code S4 corresponds to the code developed for generating the FCMacro file. *WingSegment Video S7*, in Supplementary Data, shows how the FC Macro files generated by WingSegment can be imported in FreeCAD [100].

## 5.2.3 Junction Detection

In insect wings, a junction refers to a point where two or more wing veins meet or intersect. The method for detecting wing junctions involves a series of steps. First, the width of all veins is reduced to one pixel using the Zhang-Suen thinning algorithm [81]. This process is called skeletonization, commonly used in image processing to reduce an object to its core components. In this case, skeletonization is used to reduce the width of the veins to a single pixel, making it easier to detect intersections [56, 60, 66]. The skeletonized image

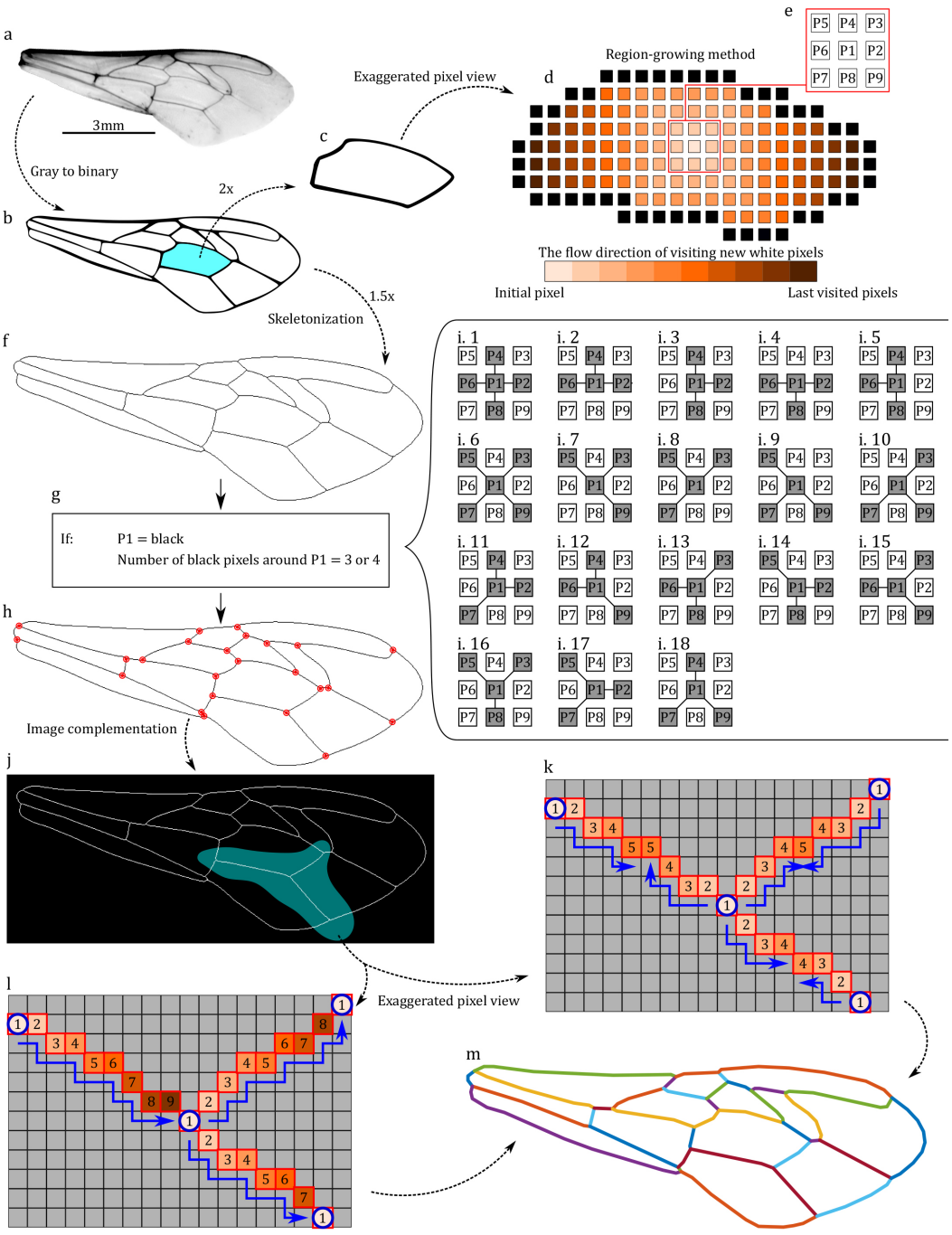



Figure 5.1: Illustration of the implemented methodology in WingSegment. a) Image of a honeybee wing, b) Binary representation of the wing image, c) Visualization of a single cell, d) Exaggerated pixel view of a single cell, e) Representation of a pixel and its surrounding pixels, f) Skeletonized image of the wing, g) Criteria for identifying junction locations, h) Detected junctions, i.1-i.18) Pixel-level depiction of the formula shown in panel g, j) Complemented skeletonized image, k) Sequential Vein Detection (SVD) method representation, l) Parallel Vein Detection (PVD) method representation, m) Detected veins in the wing.

is shown in Figure 5.1f, consisting of a network of thin lines representing the veins in the wing. To detect the junctions, the skeletonized image is examined for black pixels that are surrounded by three or four black pixels (Figure 5.1g), as shown in the conditions illustrated in Figure 5.2, i.1 to i.18. These conditions take into account the position of the black pixels relative to their neighbors, and ensure that they meet specific geometric

criteria [60, 66]. Any black pixel that meets these conditions is considered a junction. The junctions are then marked and identified in the image, as shown in Figure 5.1h. By locating the junctions in this way, it is possible to accurately map the venation patterns in the wing and obtain a detailed understanding of its structure. In the Appendices, Code S4 represents the function developed for detecting junctions.

### 5.2.4 Vein Path Extraction

We use path-finding, also known as the pathing method, to extract venation patterns here. This technique involves using a computer program to draw the most direct path between two points and is a more useful version of solving mazes. Path-finding methods rely heavily on Dijkstra's algorithm [82], which is used to find the shortest possible route on a graph with weights assigned to its edges. We have adapted this algorithm to find every path between the junctions in a wing using the following approaches.

#### Sequential Vein Detection (SVD)

This approach involves defining three sets: initial, ending, and front pixels. Each pixel in a set has a so-called parent-child relationship, where the parent indicates the origin of the child from the previous iteration. To initiate the process, one of the identified junctions is designated as the initial pixel, serving as the starting pixel for a generation (Figure 5.1l, Pixel 1). The initial pixel designates the surrounding pixels as front pixels. Since all veins in the skeletonized image are represented as white pixels, the method transfers these white pixels from the front set to the set of detected pixels. The initial pixel is considered the parent of all pixels detected afterward. Subsequently, the initial pixels are replaced by newly detected pixels, and the same procedure is repeated for each newly generated initial pixel. This process continues until a new junction is among the front pixels, indicating the detection of a vein's ending pixel. In such cases, the pixel that detects the junction and its parent begins to trace its way back to the starting pixel of its generation, with each parent returning to its own parent. This iterative process continues until they reach the initial pixel, resulting in the respective detection of all pixels associated with that vein. This procedure is then repeated for each junction, one at a time, until all veins are detected. Figure 5.1l visually illustrates this entire process. Furthermore, *WingSegment Video S5* demonstrates the detection of veins on honeybee wings using the SVD method. The code designed for implementing this method is provided and explained in detail in the Appendices under Code S7.

#### Parallel Vein Detection (PVD)

The parallel method considers all junctions simultaneously as initial pixels and begins finding paths concurrently. Consequently, as depicted in Figure 5.1k, the front pixels converge at the center of the connecting vein where two junctions are connected. *WingSegment Video S4* showcases the application of the PVD method for detecting veins on honeybee wings. The code created for this approach is presented and described in the Appendices under Code S6.

### 5.2.5 Pterostigma Detection

The Pterostigma is a feature commonly found in the wings of certain insect orders including Odonata [1, 151], Hymenoptera [152], Orthoptera [153], and Hemiptera [154].

However, the size, shape, and coloration of Pterostigma can vary among different species. To detect the Pterostigma in the wing, a set of image processing operations is applied:

- **Distance Transform.** The first step involves calculating the distance transform [155] of the wing image. The distance transform assigns to each pixel in the image a value that represents its distance to the nearest object. The Pterostigma, being a concentrated region within the wing, causes the distance transform to assign the highest value at its central location.

- **Normalization.** The resulting distance transform image is then normalized in a range of 0 to 1 using max-min normalization method. This normalization step ensures that the maximum distance has a value of 1, facilitating subsequent thresholding.

- **Thresholding.** It has been observed that a threshold of 0.7 applies to the normalized distance transform image. Pixels with values above this threshold are set to 1, while those below or equal to the threshold are set to 0. This thresholding operation helped isolate regions in the image that are likely to contain the Pterostigma.

- **Overlaying.** Finally, the threshold image is added to the original image to visually enhance the detected Pterostigma. This overlaying operation helped highlight the location of the Pterostigma by visually merging the threshold regions with the original wing image.

Following these steps, the code successfully detects and localizes the Pterostigma in the wing image.

### 5.2.6 Line Simplification

The Ramer-Douglas-Peucker algorithm, also known as the Douglas-Peucker algorithm and iterative end-point fit algorithm, simplifies a curve composed of line segments. Its purpose is to reduce the number of points while maintaining the overall shape. Originally developed for cartographic generalization, this algorithm aims to create a similar curve using fewer points. In the context of WingSegment, where both the region growing and path-finding methods detect all pixels, including many unnecessary ones, the Ramer-Douglas-Peucker algorithm is employed. By integrating this algorithm, WingSegment can eliminate redundant points and retain the critical ones that contribute to preserving the entire shape of the detected boundaries.

### 5.2.7 Graphical User Interface

WingSegment is accompanied by an accessible, user-friendly graphical user interface (GUI). The GUI is developed using MATLAB App-Designer and can be accessed in two formats via the Supplementary Data: an executable file (*.exe) for installation as standalone software on Windows desktops and a *.mlappinstall file for installation as an app within MATLAB. The developed GUI enables users to import wing images and utilize various tools such as cell and vein detection, junction identification, FreeCAD export functionality, and statistical analysis visualizations. Figure S8 shows a detailed visual depiction of WingSegment's GUI. Moreover, in *WingSegment Video S6* in Supplementary Data, we demonstrate how to use WingSegment from start to finish [100].

### 5.2.8 Validation

The performance of WingSegment was rigorously validated through comparisons with established software tools and manual measurements. First, we utilized ImageJ, a widely recognized software, for the measurement of cells' area, length, and circularity. Additionally, ImageJ was employed to measure the length of veins. However, ImageJ automatically detects the cell boundaries and measures the area of the domain. Nevertheless, the measurement of vein length, cell length, and cell circularity was conducted manually. These measurements were then compared to the corresponding results obtained from WingSegment, ensuring the accuracy and reliability of our method. Moreover, we manually measured the number of cells and junctions in a Scarlet Dragonfly wing, which served as a reference for comparison. To achieve this, we developed a custom code (available in Appendices as Code S20) that enables users to import a wing image and mark junction or cell locations with a simple click. This code facilitated the quantification of cells and junctions in a Scarlet Dragonfly. *WingSegment Video S1* and *WingSegment Video S2* in Supplementary Data respectively shows our manual counting of cells and junctions using Code S20 [100]. This allowed us to evaluate the automation capabilities of WingSegment, as it automatically extracted and quantified these features.

## 5.3 Result

Our method successfully extracts the boundary of the wing outline, individual cells, and the intricate venation pattern. Additionally, it identifies the locations of junctions and generates a 3D-printed wing model. Moreover, our method could measure the wing's geometric properties, including cell area, length, and circularity distributions, which are visualized through heatmaps, boxplots, and histograms. Furthermore, we generate a Voronoi pattern based on the centroids of the segmented wing cells. The results are summarized and visually presented in Figure 5.2, which showcases several panels. Figure 5.2a displays the extracted cells, while Figure 5.2b illustrates the precise locations of the junctions. Figure 5.2c showcases the intricate venation patterns, with each vein color-coded according to its length, as indicated by the accompanying color bar.

Figure 5.2d provides insights into the distribution of cell areas, including a distribution plot (d.1), a corresponding histogram (d.2), and a boxplot (d.3). Similarly, panels (e) and (f) present the distributions of cell lengths and circularity, respectively, with their respective histograms and boxplots. Figure 5.2g illustrates a physical model of the segmented wing created through 3D printing. The model was generated in FC using the sketch obtained from WingSegment's FC macro export tool. Printing was carried out using a Prusa MK3S printer with a 0.25 mm nozzle and Prusa Jet black PLA filament. The file containing the 3D-printable model is provided as *WingSegment-Model S1* in the Supplementary Data [100]. Finally, Figure 5.2h presents the Voronoi pattern generated based on the centroids of the extracted wing cells.

### 5.3.1 Validation

The validation of WingSegment was conducted to assess the accuracy and reliability of the method in segmenting insect wing images and extracting vital features. Tables 5.1 and 5.2 present the results obtained during the validation process, demonstrating the performance of WingSegment. Figure 5.2h illustrates the numbered cells and five randomly selected veins represented by red dashed lines. The result regarding the length of all veins is

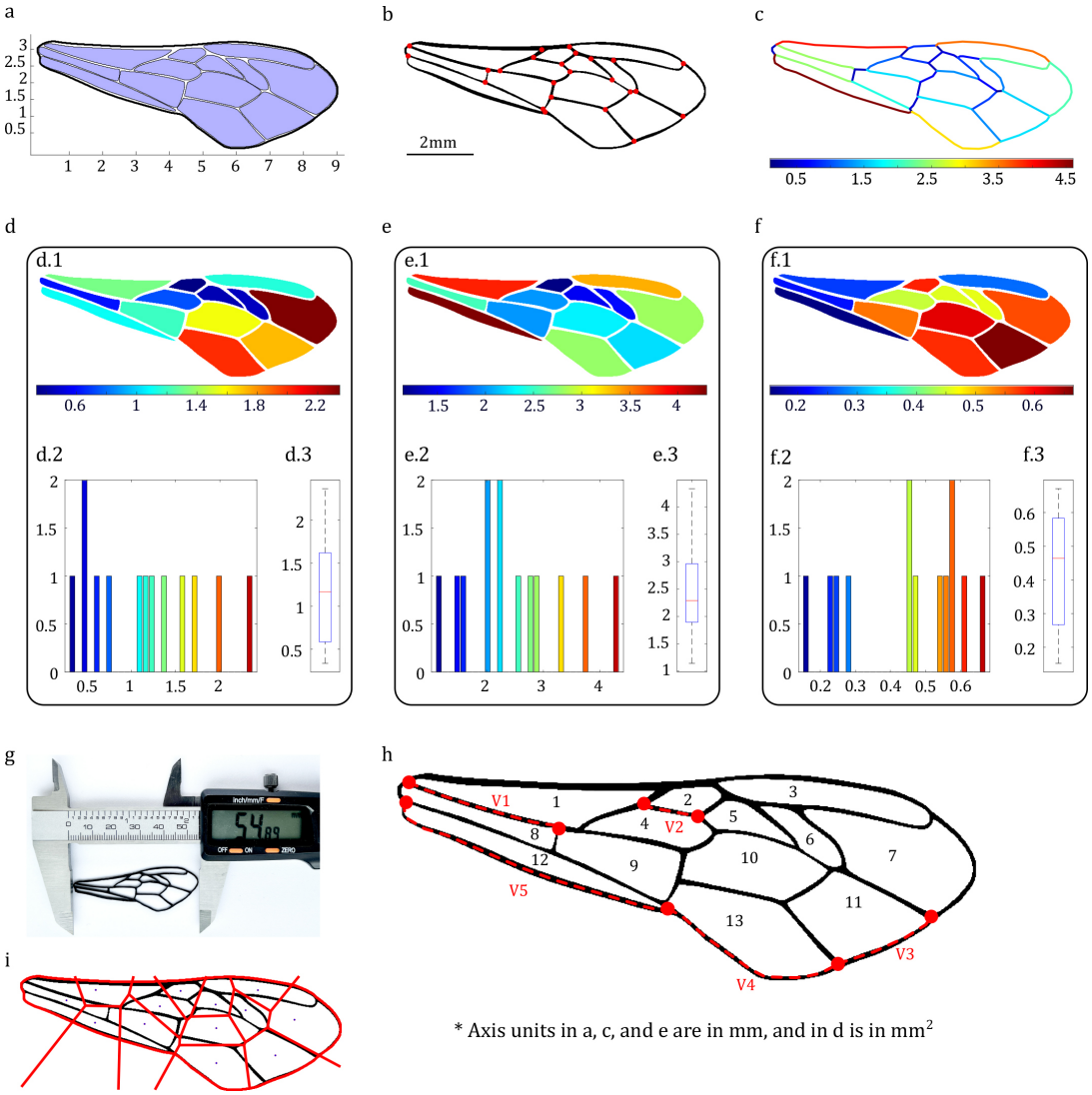



Figure 5.2: Visualization of the results obtained using WingSegment on a honeybee wing. a) Identified cells, b) Detected junctions, c) Distribution of vein lengths, d-e-f) Distribution of cell areas, lengths, and circularity. Each Figure panel consists of three subpanels: Panel 1 displays the heatmap, Panel 2 shows the corresponding histogram, and Panel 3 presents the boxplot. g) The 3D-printed physical wing model from the 3D model was generated using WingSegment. h) Labeled cells and veins corresponding with the results of Tables 5.1 and 5.2 i) Generated Voronoi pattern.

stored as *WingSegment-File S5* in Supplementary Data [100]. The subsequent analysis focused on comparing results between WingSegment and ImageJ. Table 5.1 provides the vein-related results, revealing an error rate of only 0.56%. This low error percentage underscores the precision of our method in detecting and measuring the intricate venation patterns present in insect wings. Moving on to cell analysis, Table 5.2 presents the obtained cell area measurements, circularity, and length from both WingSegment and ImageJ. The results indicate a high accuracy of cell area measurements, with negligible differences observed. The average error in cell circularity and length measurement is 3.70% and 1.59%, respectively.

Moreover, manual counting was performed on a complex Scarlet Dragonfly forewing to assess the cell and junction counting functionality of WingSegment. *WingSegment Videos S1* and *WingSegment Videos S2* in Supplementary Data illustrate the manual marking process for cells and junctions using our custom code (Code S20 in Appendices), respec-

tively. The coordinates of cells' centroids and junctions are stored as *WingSegment-File S1* to *WingSegment-File S4* in Supplementary Data [100]. Remarkably, both methods yield the same count for cells (440) and junctions (816), highlighting the exceptional performance of WingSegment in accurately detecting cells and junctions.

| | **WingSegment** | **ImageJ** | |
|---|---|---|---|
| | Vein Length (mm) | Vein Length (mm) | Error % |
| Vein 1 | 2.35 | 2.33 | 0.77 |
| Vein 2 | 0.84 | 0.83 | 0.72 |
| Vein 3 | 1.57 | 1.56 | 0.45 |
| Vein 4 | 2.97 | 2.95 | 0.68 |
| Vein 5 | 4.61 | 4.60 | 0.17 |
| | | Average% | 0.56 |

Table 5.1: Comparison of measured vein lengths using WingSegment and ImageJ, along with the computed average error.

| | **WingSegment** | | | **ImageJ** | | | **Error%** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Area* | Circularity | Length* | Area* | Circularity | Length* | Area* | Circularity | Length* |
| Cell 01 | 6345 | 0.23 | 268.27 | 6348 | 0.24 | 266.71 | 0.05 | 2.10 | 0.59 |
| Cell 02 | 1547 | 0.53 | 81.74 | 1547 | 0.57 | 79.46 | 0.00 | 6.37 | 2.87 |
| Cell 03 | 5540 | 0.26 | 235.60 | 5543 | 0.27 | 234.03 | 0.05 | 2.58 | 0.67 |
| Cell 04 | 3561 | 0.42 | 141.17 | 3563 | 0.44 | 140.01 | 0.06 | 4.08 | 0.83 |
| Cell 05 | 2097 | 0.43 | 106.25 | 2100 | 0.45 | 103.92 | 0.14 | 5.76 | 2.24 |
| Cell 06 | 2115 | 0.42 | 111.61 | 2116 | 0.45 | 108.63 | 0.05 | 5.61 | 2.74 |
| Cell 07 | 11343 | 0.55 | 199.46 | 11347 | 0.57 | 199.74 | 0.04 | 3.00 | 0.14 |
| Cell 08 | 2876 | 0.21 | 178.84 | 2878 | 0.22 | 174.44 | 0.07 | 2.78 | 2.52 |
| Cell 09 | 5757 | 0.53 | 141.03 | 5760 | 0.55 | 137.06 | 0.05 | 3.83 | 2.90 |
| Cell 10 | 7558 | 0.60 | 160.38 | 7564 | 0.62 | 159.70 | 0.08 | 3.87 | 0.42 |
| Cell 11 | 8302 | 0.65 | 157.28 | 8303 | 0.67 | 154.11 | 0.01 | 3.15 | 2.05 |
| Cell 12 | 5071 | 0.14 | 304.36 | 5076 | 0.15 | 301.94 | 0.10 | 2.07 | 0.80 |
| Cell 13 | 9659 | 0.56 | 196.87 | 9662 | 0.58 | 193.30 | 0.03 | 2.93 | 1.84 |
| | | | | Average % | | | 0.056 | 3.70 | 1.59 |

* In this table, the units of area and length are $px^2$ and $px$, respectively.

Table 5.2: Comparison of measured cell area, circularity, and length using WingSegment and ImageJ, alongside the computed error.

## 5.4 Discussion

This study suggests a new hybrid method called WingSegment to segment insect wing images. The methodology employed in WingSegment combines region growing and thinning algorithms for detecting cells and junctions, along with path-finding techniques for extracting venation patterns. This hybrid approach ensures efficient and accurate analysis of complex wing structures. It is important to emphasize that integrating the aforementioned algorithms is crucial in this case, and considerable effort is required when employing these algorithms separately for the same task. In WingSegment, the embedded method extracts the locations of junctions by utilizing the thinning algorithm and the conditions introduced in the methodology. The pathfinding method then determines the path between the extracted junctions from the previous step. Additionally, the skeletonized image enables the algorithm to identify the available networks between junctions and uncover the true path, even if it is curved. This capability is not present in currently available tools like NET and NEFI. Performance evaluation was conducted on honeybee and *Crocothemis erythraea* (Scarlet dragonfly, Odonata: Anisoptera; hindwing) to assess the effectiveness of WingSegment (Figures 5.2, and 5.3). In addition, additional figures

corresponding to the geometric features of a damselfly, a locust wing, and a Scarlet Dragonfly forewing are presented in Figures S10 to S13 in the Appendices. This comprehensive evaluation allowed a thorough assessment of the software's performance across different insect species.

The results obtained from WingSegment demonstrate its effectiveness, as evidenced by the successful extraction of wing features with minimal errors in vein detection and cell measurement. The WingSegment validation involved comparing the results with those obtained from established tools such as ImageJ and manual measurements. The results obtained from WingSegment were in agreement with these reference methods, confirming the accuracy and reliability of our approach. It is important to note that both WingSegment and ImageJ employ image binarization as a preprocessing step. In WingSegment, we set the threshold value to 0.54, which provides optimal alignment with ImageJ results. Altering this threshold value would impact the outcome.

### 5.4.1 WingSegment vs. WingGram

WingSegment, while visually resembling its counterpart WingGram [60], incorporates notable distinctions and numerous enhancements. Both tools employ a similar approach to extract cell boundaries, but WingSegment addresses minor bugs from WingGram, resulting in enhanced reliability and speed. Figure 5.3a illustrates the extracted cells from a dragonfly wing, represented in blue color. While both WingGram and WingSegment provide measurements for cell area, length, and count, WingGram includes an additional feature that measures cell width by dividing the area by the length of cells. However, this approach was found to be imprecise and has been omitted from WingSegment. Instead, WingSegment incorporates cell circularity measurement, providing valuable information about cell shape [43, 55]. Figures 5.3f, g, and h depict the distribution of cell area, length, and circularity using various visual formats, including heatmap (Panel 1), histogram (Panel 2), and boxplot (Panel 3). Panel 1 provides an overview of the distribution of cells and offers insights into their location based on area, length, or circularity. This type of figure enables the examination of the relationship between cell size and its spatial positioning. The histogram in Panel 2 provides valuable information on how cells are distributed across different categories of area, length, and circularity. This allows for investigating the predominant size or shape of the cells within the population. Lastly, the boxplot in Panel 3 presents important statistical details such as the minimum, maximum, upper quartile, lower quartile, and median values of cell area, length, and circularity. These visualizations collectively provide insights into the characteristics and distribution of cells within the wing sample.

Furthermore, WingSegment offers additional features not present in WingGram. It provides distance measurements from the wing margins, allowing for a more comprehensive analysis of the wing structure. Additionally, WingSegment includes automatic detection of the Pterostigma, a specialized region within the wing. In terms of junctions, both tools are capable of locating and quantifying junctions. However, WingSegment surpasses WingGram by identifying the ends of unblocked veins as junctions, which provides a more accurate representation of the wing's connectivity. Notably, both WingGram and WingSegment can detect cell centroids on insect wings. However, WingSegment stands out with its built-in feature for generating a Voronoi pattern based on the extracted centroids. Figure 5.3b illustrates the Voronoi pattern generated by WingSegment on a dragonfly wing. This Voronoi pattern is a significant aspect of WingSegment, as it offers insights into the arrangement and distribution of cells on the wing surface [55].

### 5.4.2 WingSegment vs. WingMesh

WingMesh is another software developed for mesh generation on insect wing geometries [59]. It was created by integrating image processing with distmesh2d [99], a well-known mesh generator in MATLAB. In WingMesh, the wing's geometry is extracted using image analysis, and distmesh2d is then employed to generate a mesh on the extracted geometry. The output mesh in WingMesh is based on INP files compatible with Abaqus CAE. While WingMesh represents an advancement in automated mesh generation for insect wing geometries, it is noteworthy that it generates orphan meshes in Abaqus. Consequently, manipulating the geometry on the finite element (FE) models generated by WingMesh is impossible. Moreover, WingMesh was slow, particularly demonstrating reduced functionality on wings with complex geometries, such as dragonfly wings. Table 5.3 provides a side-by-side comparison illustrating the key differences between WingGram, WingMesh, and WingSegment.

| Tools and Options | WingSegment | WingGram | WingMesh |
|---|---|---|---|
| Abaqus INP Model | No | No | Yes |
| Abaqus JNL Model | No | Yes | No |
| FreeCAD Model | Yes | No | No |
| Cells' Area | Yes | Yes | No |
| Cells' Length | Yes | Yes | No |
| Cells' Width | No | Yes | No |
| Cells' Circualrity | Yes | No | No |
| Cells' Centorids | Yes | Yes | No |
| Cells' Distance from Margin | Yes | No | No |
| Voronoi Pattern Generation | Yes | No | No |
| Junction Detection | Yes* | Yes | No |
| Venation Pattern Detection | Yes | No | No |
| Pterostigma Detection | Yes | No | No |
| Applicable for Complex Geometries | Yes | Yes | No |
| 3D Printable Model Generation | Yes | Yes | No |
| Corrugation Assignment | No | Yes | Yes |

*Improved and Modified

Table 5.3: Comparison between WingSegment, WingGram, and WingMesh

### 5.4.3 Venation Pattern Recognition

One distinguishing feature of WingSegment, which was not incorporated in WingGram, is its ability to extract the network pattern of insect wings and accurately determine the paths between junctions. While tools like NEFI [66] and NET [65] excel in extracting network patterns from various structures such as leaves and spider webs, WingSegment offers a unique advantage in capturing the intricate details of wing venation. NEFI and NET primarily focus on network extraction, representing connections between junctions with straight lines, which is suitable for their intended purposes as network analysis tools. However, WingSegment goes beyond network extraction by precisely capturing the curved paths between junctions, allowing for a more comprehensive representation of the intricate venation pattern and providing valuable insights into the wing's structural characteristics. It is important to highlight the specific focus and strengths of NEFI and NET in network analysis while emphasizing that WingSegment complements these tools by offering a specialized feature for path extraction.

Figure 5.3a demonstrates the successful extraction of the intricate venation pattern from a complex dragonfly wing represented by red color. Notably, when dealing with a dragonfly wing featuring a Pterostigma, Figure 5.3a.2 showcases the effectiveness of the distance transform in detecting veins around the Pterostigma. In contrast, Figure 5.3a.3 illustrates the detection of a vein within the Pterostigma when this option is not utilized. This function enhances vein detection and leads to more accurate junction detection. Furthermore, Figure 5.3c provides insight into the distribution of veins based on their length.

Additionally, this paper presents two approaches for detecting venation patterns: Sequential Vein Detection (SVD) (*WingSegment Video S5* [100]) and Parallel Vein Detection (PVD) (*WingSegment Video S4* [100]). While PVD offers faster processing due to its parallel nature, it is essential to note that accurately extracting venation patterns is a highly delicate process. Initially, we developed SVD and subsequently enhanced its speed. Following that, we developed PVD. Our observations revealed that while both methods generally detect all veins within the wing, there are instances where one of the methods may miss some veins. Consequently, we incorporated both methods into WingSegment to ensure comprehensive vein detection.

### 5.4.4 FreeCAD Macro Export: Seamless Transfer of the Wing Geometry to FreeCAD

In terms of finite element Modeling (FEM), WingGram surpasses WingSegment with its array of tools for modeling insect wings. WingGram offers functionalities such as marking discontinuities to exclude specific areas from the model and an innovative tool for corrugation assignment. Furthermore, WingGram can export the final model as an Abaqus JNL file. On the other hand, WingSegment generates a FreeCAD macro file from extracted cell boundaries, allowing for the import of a wing cell sketch into FreeCAD. However, it is essential to consider that Abaqus is not open access, limiting accessibility for many users. In contrast, the FC macro exporting tool in WingSegment, although lacking corrugation assignment capabilities, generates a file compatible with FreeCAD—an open-source and accessible FEM software package. This aspect enhances the usability and practicality of WingSegment, particularly for users without access to specialized software like Abaqus. Furthermore, this seamless integration empowers users to easily create FE or 3D printable models within the FreeCAD environment. The practical utility of WingSegment is exemplified in Figure 5.3e, showcasing a successful application of 3D printing. More examples of dragonfly, damselfly, and locust wings with complex geometries are also available in Figure S9 in the Appendices. The physical wing was produced using a Prusa MK3S 3D printer, employing a 0.25 mm nozzle size. Prusament PLA Jet black filament was used to fabricate the wing in Figure 5.3e, and Prusament PLA Galaxy Silver was used in producing wings in Figure S9. The nozzle temperature was set at a constant value of 225℃, while the bed temperature was maintained at 65℃. To ensure precise replication of all features, the wing model was scaled approximately 4.5 times its original size, resulting in a 3D-printed wing that accurately represents the morphology of the original insect wing. The 3D printable model of the wing is stored as *WingSegment-Model S2* in Supplementary Data [100].

It is essential to highlight that, like WingGram, WingSegment extracts all points along the boundaries of cells, which can lead to many unnecessary points. These plenty of points significantly increase the volume of the macro file, consequently increasing the runtime during the importing process in FreeCAD. Therefore, it is crucial to recognize

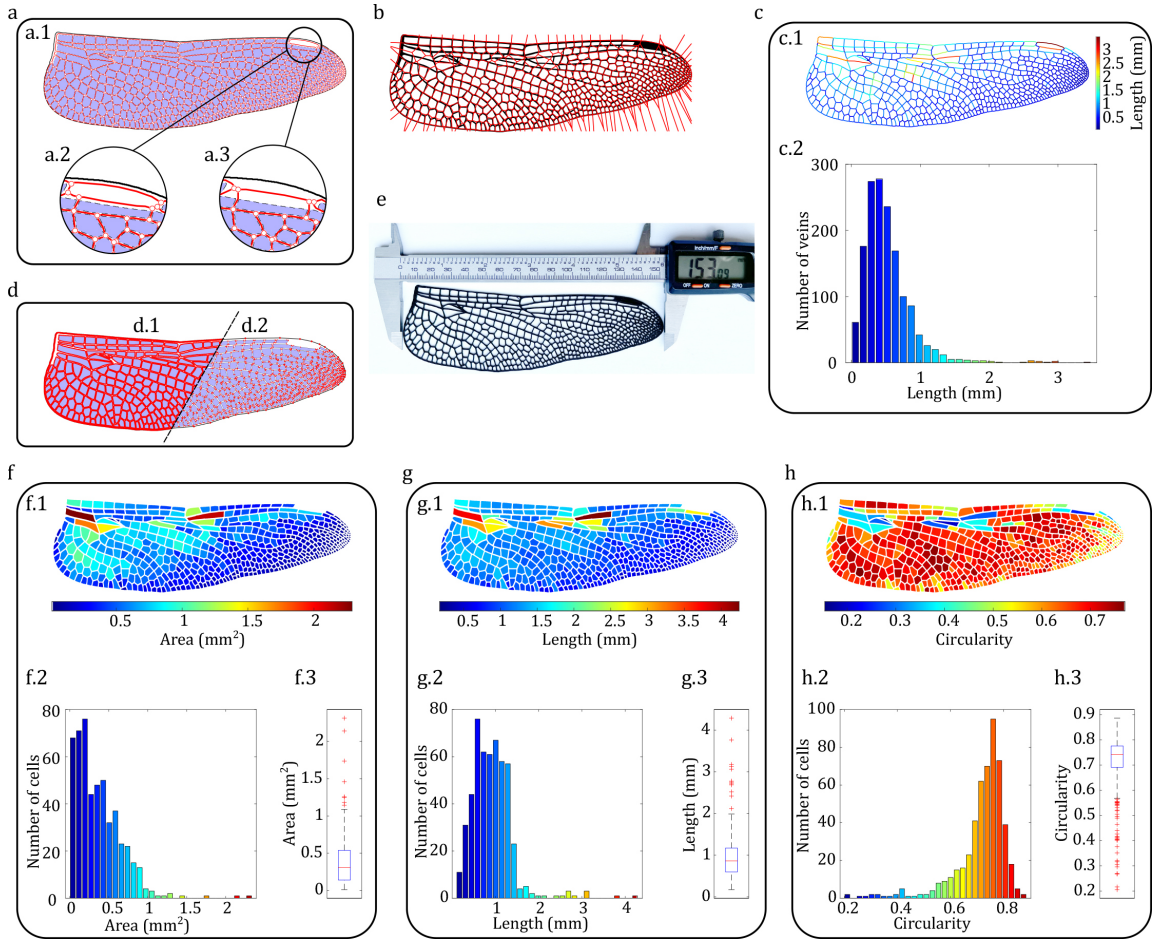



Figure 5.3: Evaluation of WingSegment performance on a complex dragonfly wing. a) Extraction of cells, junctions, and venation patterns. Panels 2 and 3 demonstrate the impact of using and not using the distance transform function in Pterostigma detection, respectively. b) Generated Voronoi pattern based on extracted cell centroids. c) Visualization of vein length distribution. Panel 1 displays the heatmap, and panel 2 presents the histogram. d) Illustration of line simplification technique. Panels 1 and 2 depict point density before and after applying the line simplification method. e) Three-dimensional printed dragonfly wing. f-g-h) Analysis of cell area, length, and circularity, respectively. Panels 1, 2, and 3 show the heatmap, histogram, and boxplot.

the importance of employing line simplification techniques before generating the macro file. This is exemplified in Figure 5.3d, where the density of points before and after line simplification is showcased in Figure 5.3d.1 and Figure 5.3d.2, respectively.

### 5.4.5 Robustness of WingSegment

WingSegment demonstrates exceptional versatility in segmenting wings with both simple and complex geometries. This capability is exemplified through the segmentation of a Honeybee wing (Figure 5.2) and the intricate hindwing of a Scarlet Dragonfly (Figure 5.3). Furthermore, to validate the robustness of cell and junction extraction, we utilized a Scarlet Dragonfly forewing, showcasing the software's performance (Figure S6). We have also presented additional 3D printed models of wings with a high number of cells and geometric complexity in Figure S9 of the Appendices. These wings are from the Damselfly, Locust, and Scarlet Dragonfly forewings.

In practice, WingSegment adeptly handles various wing complexities but is more sensitive to image quality. While very high-resolution images may impact runtime, excessively

low-resolution images can affect software performance. In our assessments, images with a resolution of 300 dpi were found optimal.

It is important to note that WingSegment excels when the wing image is clear. The software successfully extracts features, but the user must ensure that the image quality meets certain criteria. Images with thin veins faded details, or noise may not yield optimal results. WingSegment distinguishes between black and white pixels, crucial in cell and vein detection tasks. However, this feature has limitations; for instance, black pixels are considered veins in the region growing method. Therefore, WingSegment is particularly well-suited for wings with transparent membranes and dark veins. Challenges arise when dealing with transparent veins, necessitating user intervention to enhance image contrast.

It's worth mentioning that WingSegment may not be suitable for wings with dark colors or black spots due to its pixel differentiation method. Nonetheless, any wing image can be pre-processed to meet the software's requirements. To address these limitations, we are actively developing filters and incorporating deep learning and machine learning methodologies to enhance software flexibility, minimizing the need for manual editing or pre-processing. Our ongoing efforts include integrating venation sections and corrugations into final 3D models generated with WingSegment.

### 5.4.6 Applications of WingSegment

WingSegment may play a pivotal role in advancing the study of insect wings by offering automated solutions, significantly reducing the effort required for analyzing mass samples. The software's ability to rely solely on wing images without additional specialized information enhances its accessibility and applicability. WingSegment may contribute substantially to understanding the asymmetry of insect wings, extracting various geometric features from images [156]. Moreover, it will facilitate the design and development of bio-inspired and biomimetic wings by generating intricate 3D models [21, 157]. The tool also may aid in studying the functional morphology of wing venation patterns, cells, and junctions through automated extraction of these elements from wing images. Beyond entomology, scientists can harness WingSegment for feature extraction in machine learning and deep learning applications, enabling the development of automated identification, modeling, and analysis methods for insect wings [67–70]. Notably, WingSegment's versatility extends beyond insect wings, as it can extract features from 2D images of other cellular structures such as leaves, skins, and foam-like formations like that illustrated in Figure 5.4. Additionally, as depicted in Figure 5.4b, and c, it can contribute to the investigation of soil and the development of clay cracks [158]. This broadens its impact on diverse fields, including developing and designing cellular thin films, Geocells, and modeling and analyzing leaves and plant species determination. We envision WingSegment as an invaluable tool in entomology and the comprehensive study of various planar biological structures. WingSegment showcases advancements in designing and developing 3D modeling and printing of wings and exhibits their potential applicability to a wide array of planar structures. This can contribute to advancing 3D printing technologies [159].

## 5.5 Conclusion

In conclusion, this paper introduces WingSegment, a novel computer vision-based hybrid method for insect wing segmentation, venation extraction, and FreeCAD modeling. By leveraging the advantages of computer vision, graph theory, and image analysis,

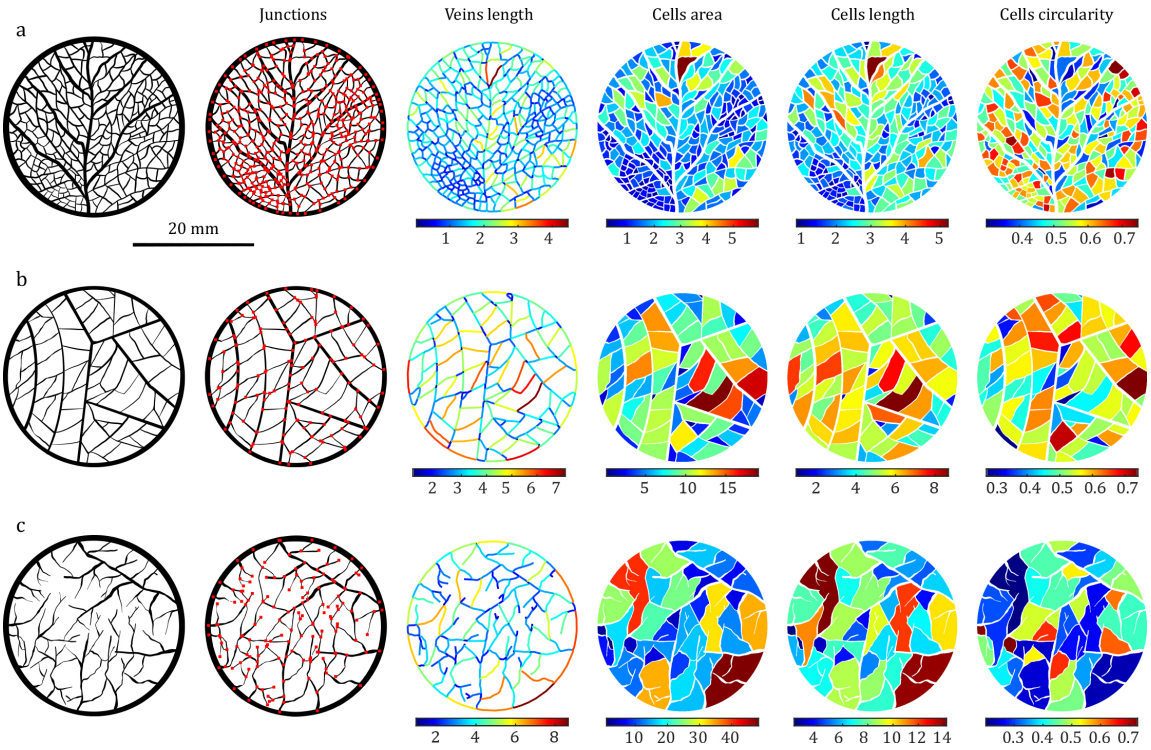



Figure 5.4: Illustration of WingSegment performance beyond insect wings. a) Leaf vein pattern. b,c) Soil crack pattern

WingSegment offers a powerful solution for studying and analyzing complex insect wings. The proposed method demonstrates remarkable accuracy in extracting cell boundaries, precisely detecting vein patterns, and capturing junctions. Additionally, WingSegment provides various geometric statistical features, such as cell area, length, and circularity. Moreover, the integration of a user-friendly GUI and the generation of FreeCAD macro files enhance the accessibility and usability of the software.

The experimental results, showcasing the application of WingSegment on a honeybee wing, highlight the successful extraction of cells and veins and the subsequent creation of a detailed 3D model. Validation using ImageJ software further confirms the reliability and effectiveness of WingSegment. Furthermore, WingSegment and the previously developed tools, WingGram and WingMesh, are comparatively discussed (Table 5.1). While WingSegment surpasses WingGram in venation pattern extraction and circularity measurement, WingGram retains its own advantages for FE modeling of insect wings. Notably, the accessibility and affordability of WingSegment, which employs FreeCAD as a platform, make it an attractive option for students and researchers. Our development simplifies the study of insect wings and opens avenues for future developments. With its versatility, the presented software is exemplified by exploring a more complex dragonfly wing, further demonstrating the method's robustness—intricate insect wing structures, empowering researchers and students with valuable study tools.

# Chapter 6

# *Paper IV.* WingAnalogy: A Computer Vision Based Tool for Automated Insect Wing Asymmetry and Morphometric Analysis[1]


## Summary

WingAnalogy is a MATLAB-based software package with a graphical user interface tailored to analyze and compare insect wing images. This versatile tool empowers researchers to effortlessly explore intricate aspects of wing morphology and asymmetry. The software facilitates project management, enabling users to import pairs of wing images, such as left and right, fore- and hindwings, from individual insects. Following image import, WingAnalogy employs image processing and computer vision techniques to autonomously segment wing structures, extract cell boundaries, and locate junctions. It quantifies essential metrics encompassing cell and wing characteristics, including area, length, width, circularity, and centroid positions. Moreover, the software enables users to scale and superimpose wing images utilizing a Particle Swarm Optimization (PSO) approach. WingAnalogy distinguishes itself by establishing correspondences between cells in paired wings, paving the way for a comprehensive wing asymmetry assessment. It computes a suite of asymmetry metrics, encompassing regression, Normalized Root Mean Square Error (NRMSE), various cell-based parameters, and distances between cell centroids and junctions. The software generates informative visualizations, aiding researchers in comprehending and interpreting asymmetry patterns. WingAnalogy allows for dividing wings into up to five distinct wing cell sets, facilitating localized comparisons. Researchers can analyze asymmetry across multiple wing pairs within a project, fostering comprehensive insights. The software excels in report generation, providing detailed asymmetry measurements in PDF, CSV, and TXT formats. Accessible as both a standalone application on Windows and a MATLAB toolbox, WingAnalogy is freely available, promoting widespread utility within the scientific community. In summary, this software is an indispensable asset for entomologists and other researchers, offering an efficient, user-friendly platform for meticulously examining insect wing morphology.


[1]This chapter is currently in the peer-review process.

## 6.1 Introduction

With their delicate and intricate wings, insects provide a canvas for scientists and researchers to explore the mysteries of evolution, adaptation, and biodiversity [160]. The study of insect wing morphology has been a cornerstone question of entomology, offering profound insights into biomechanics, animal behavior, physiology, and ecology [1, 8, 73, 74, 109, 110, 118, 130, 161]. Researchers have used wing characters, especially venation patterns [162] for decades to reconstruct the phylogeny and evolutionary history of insects and establish diagnostic characters to differentiate species and higher taxonomic levels [163, 164]. Additionally, insect wings fascinated researchers for their vital roles in damage prevention [107], thermoregulation [3], and communication [165]. Through the examination of wing structures, researchers have revealed insights into the evolutionary adaptations, species diversity, and ecological roles of insects [43, 51, 55, 166]. While wing structure holds immense significance, researchers have faced persistent challenges in analyzing complex wing venation that have prevented their quest for precise and comprehensive results. Conventional manual techniques proved to be time-consuming, prone to human mistakes and frequently fell short of the precision demanded by difficult scientific inquiries [139, 140]. The demand for advanced, automated, and user-friendly data-driven solutions became increasingly evident, prompting the development of innovative software tools [56, 57].

Before this study, the authors established their expertise in software development by creating several computer tools for studying and analyzing insect wings. WingMesh was designed to generate mesh structures along the boundaries of insect wings, utilizing computer vision techniques with distmesh2d [58, 59, 99]. Another software, WingGram, was developed to perform finite element modeling of insect wings. This was achieved using Python scripting in Abaqus, in conjunction with computer vision methodologies [60]. The authors also introduced WingSegment [77], a software application designed to segment insect wing images into constituent elements, including cells, junctions, and veins. This process involved employing various image processing, computer vision, and graph theory techniques, such as region growing [71, 72, 79, 80], pathfinding [82], thinning [81], and line simplification [83, 84]. Furthermore, WingSegment had practical applications for generating 3D printable models of insect wings from their images, accomplished through FreeCAD Python scripting by generating macro files [147, 148].

While the mentioned software packages are primarily designed for tasks, such as extracting geometric features of insect wings, finite element analysis, or 3D modeling of the wing, they seem to lack specific tools for automated analysis of the insect wings' asymmetry. However, it is important to note that investigating wing asymmetry in insects is very important, as it provides valuable insights into various areas, including flight biomechanics, evolutionary adaptations, ecological interactions, environmental health monitoring, taxonomic identification, and the potential for applications in biomimetics and materials science [167, 168]. It is important to note that the study of insect wing asymmetry has developed significantly with technological advancements. In earlier research, manual cell counting was employed to compare specific wing areas, such as those between the nodus and pterostigma along the coastal margin in damselflies [46]. Subsequent studies expanded their analysis to include wing length measurements alongside manual cell counting in damselflies [47]. For a more in-depth examination of wing three-dimensional structures, micro-CT scanners have been utilized [48]. However, while meticulous, this method is too complex and time-consuming for analyzing many individuals. Another research introduced automated software to count cells in photographed wings, streamlining the process [49]. Moreover, in another study, researchers combined data from various

Odonata species to create a predictive model for wing venation, utilizing both scanned and photographed wings [55]. Meanwhile, other researches adopted landmark-based techniques to compare the overall shape of specific wing areas [50–54].

In this paper, we aim to address the lack of previous methods regarding manual analysis or landmark-based methods in studying the asymmetry of insect wings by introducing a new software, WingAnalogy. Featuring a user-friendly graphical interface, WingAnalogy assists a broader audience, including scientists without programming expertise. The design of WingAnalogy draws upon a diverse array of algorithms encompassing computer vision, image processing, statistics, and metaheuristics, all comprehensively detailed within this paper. We conducted manual measurements using ImageJ to validate the accuracy of WingAnalogy's results. Case study species were chosen to represent wings of increasing complexity (number of cells and junctions): *Apis mellifera* (Honeybee, Hymenoptera; forewings), *Ischnura elegans* (Blue-tailed damselfly, Odonata: Zygoptera; forewings), *Crocothemis erythraea* (Scarlet dragonfly, Odonata: Anisoptera; forewings), and *Schistocerca gregaria* (Desert locust, Orthoptera; hindwings). In the subsequent sections, we delve into the advancements brought forth by WingAnalogy while also addressing its limitations.

## 6.2 Methods

WingAnalogy, developed using MATLAB App Designer (Version 2020b), is a software tool that streamlines and simplifies the computation of asymmetry in pairs of insect wings. This tool assesses wing asymmetry based on 2D wing images. The criteria for evaluating wing asymmetry encompass regression and normalized root mean square error calculations involving parameters such as area, length, width, and circularity of corresponding wing cells, the distances between junctions and outlines, and the number of cells and junctions. Upon importing a wing image, the integrated algorithm within WingAnalogy segments the image by extracting the wing outline and cells' boundaries and identifying the junction locations. Once the image is segmented, users can scale the wing. The subsequent step involves superimposing the imported wings, which are carried out automatically or manually. Subsequently, the embedded functionality in the software measures the extent of asymmetry in the wings. Users can divide the wing into a maximum of five cell sets during this stage, allowing for localized comparisons between different wing cell sets. A noteworthy feature of WingAnalogy is its capability to import multiple wing pairs (forewing, hindwings within a species, or specific pairs of wings in different species), enabling users to comprehensively compare the asymmetry results across all wing pairs and wing pair sets. Upon completing asymmetry calculations, users can generate a TXT and a CSV file containing comprehensive asymmetry data and multiple Figures detailing cell attributes (area, width, length, circularity) and their regression analyses between paired wings. The user can also generate a PDF report of all measurements and analyses.

### 6.2.1 Image Preparation

Ensuring optimal image quality is important because it directly influences data extraction accuracy. Additionally, both comparison images must share the same dimensions and resolution. Images can be obtained through a camera or scanner. As demonstrated in Figure 6.1a,b, we illustrate the image of the left and right damselfly wing obtained using a high-resolution scanner (Epson Perfection V850 Pro). One crucial guideline in

WingAnalogy is to ensure that all images are oriented in the same direction. Therefore, as shown in Figure 6.1f, it is required to mirror the left wing to align it with the right wing before conducting wing analysis. Users must ensure that all wing veins and membranes remain distinctly visible after the binarization process by converting the image into a binary representation consisting solely of black and white pixels determined by a threshold value of 0.54. For reference, Figure 6.1a and b are suitable examples of an image that aligns with the prerequisites for effective utilization within the WingAnalogy functionality. In addition, *WingAnalogy-Sample Image S1* to *WingAnalogy-Sample Image S8* in the Supplementary Data [100] are images specifically chosen for analysis in this study. Furthermore, Code S2 in the Appendices is the code developed to import wing images in WingAnalogy. *WingAnalogy Video S1* in the Supplementary Data demonstrates the adequate preparation of a wing image [100]. While manual preparation can improve results, it is not always required. Video S1 was designed to demonstrate the worst-case scenario, whereas simpler edits are typically adequate.

### 6.2.2 Image Segmentation

Upon importation, every image undergoes an automated segmentation process formulated according to computer vision principles, and image processing detects all boundaries within the wing. In this approach, a region growing algorithm is employed to extract cell boundaries [71, 72, 79, 80]. Figure 6.1d provides an exaggerated view of the cell highlighted in Figure 6.1c, which is selected from the right wing of the damselfly depicted in Figure 6.1b. The region growing method needs an initial point, such as pixel number 1 in Figure 1d. As demonstrated in Figure 6.1e, a random pixel, like p1, is surrounded by eight neighboring pixels (P2-P9). Subsequently, the initial pixel 1 examines all surrounding pixels. A white pixel is saved as the front pixel as the method proceeds its search using front pixels. Conversely, if a pixel is black, the algorithm records its coordinates as the boundary of that particular region. During each iteration, the algorithm changes the color of each detected pixel to prevent redundancy. This process persists until all white pixels within the boundary have been examined. At this stage, all boundary pixels are identified. The algorithm then proceeds to another region and repeats the same procedure. As a result, the region growing method ultimately identifies all cell boundaries within the wing. Figure 6.1h displays the detected boundaries of the image in Figure 6.1b, achieved through the region growing method. After boundary extraction, the number of detected white pixels inside each region represents the area of that cell. The maximum distance between the boundary of each cell represents the length. Width is measured by dividing the area by the length, and the circularity is measured using Equation 6.1:

$$C = (4\pi * area)/(perimeter^2) \tag{6.1}$$

Where *area* is the area of the cell, and the *perimeter* is the perimeter of the cell measured by the region growing algorithm.

All measurements about wing size and its constituent cells are conducted in pixel units. Consequently, an integrated feature within WingAnalogy enables scaling all measurements. Figure S17 shows the distribution of cells area, length, width, and circularity and corresponding histograms of the wing in Figure 6.1b. Moreover, WingAnalogy employs a thinning algorithm to skeletonize imported images [81]. Figure 6.1g illustrates the skeletonized version of Figure 6.1f. Within the skeletonized image, any black pixel surrounded by 3 or 4 other black pixels (essentially, any pixel meeting one of the 18 conditions outlined in Figure 6.1i), is identified as a junction. Figure 6.1j presents all

the detected junctions on the wing. Moreover, Code S3, Code S4, and Code S5 in the Appendices are developed for the region growing method, wing segmentation, and wing junction detection.

### 6.2.3 Superimposing

The primary goal of superimposition using the Procrustes method is to eliminate differences in position, orientation, and scale between shapes so that they can be more easily compared and analyzed [169]. Superimposing in the context of comparing two images refers to overlaying or combining two images on top of each other to analyze their differences and similarities visually. In WingAnalogy, we skip scaling between shapes because we assume that the images used are captured at the same size. This is because, during wing comparisons, there might be instances where one wing is larger or smaller, and this size difference could be an essential criterion for comparison [170]. In the WingAnalogy software, this step follows the automatic extraction of geometry and scaling data and serves as the initial phase for comparing insect wings. The superimposing process can be carried out either manually or automatically.

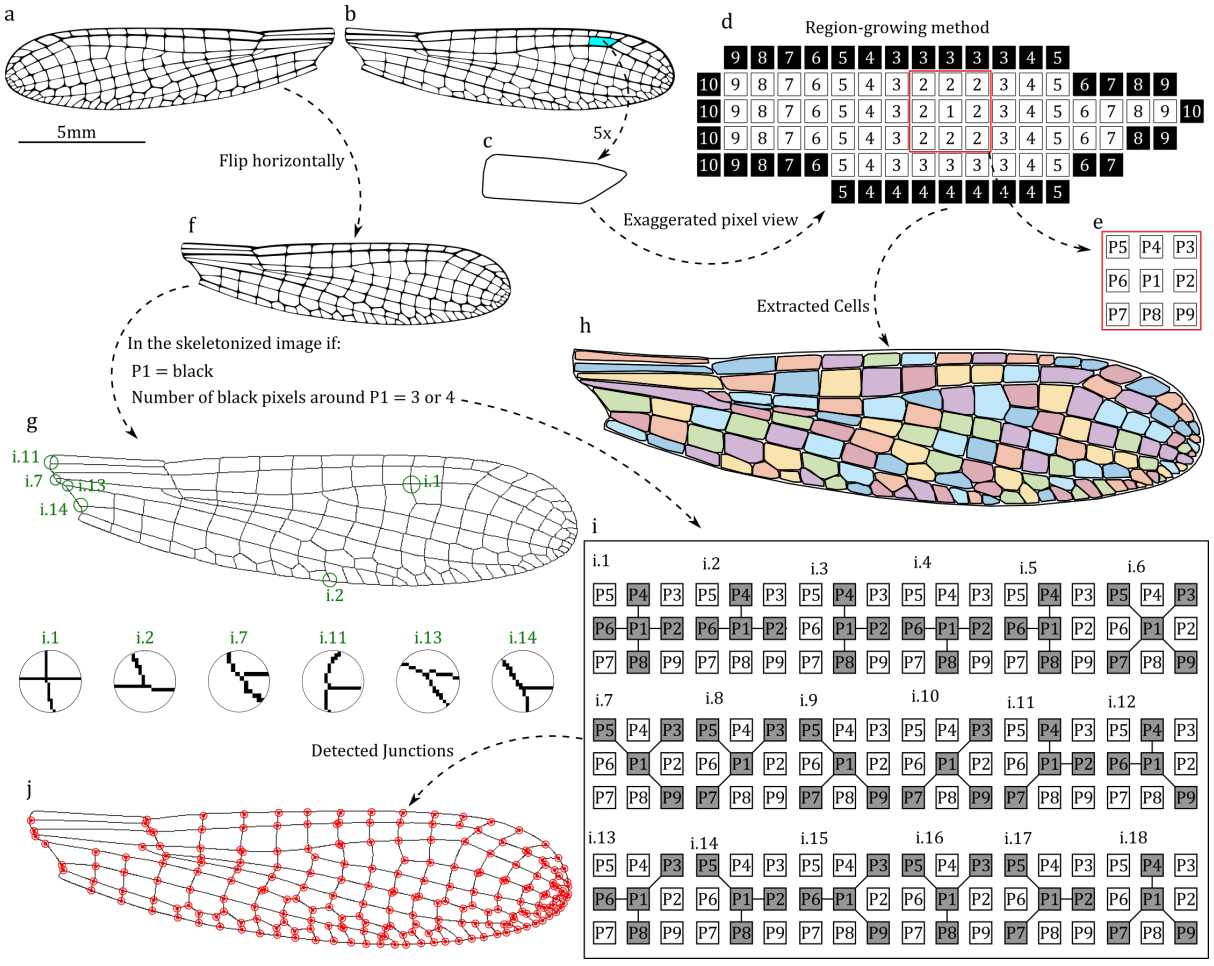



Figure 6.1: Illustration of the WingAnalogy segmentation process. (a) and (b) The left and right wings of damselfly wings, respectively. (c) A selected cell within the right wing. (d) An exaggerated view of the cell is highlighted in (c). (e) A pixel with its surrounding pixels. (f) The mirrored image of the left wing from (a). (g) The skeletonized image of the wing from (f). (i) The conditions that, if met, classify a pixel as a junction. (j) The detected junctions within the wing.

### Manual Superimposing

For manual superimposition, the software incorporates a set of ten buttons illustrated in Figure S18 that facilitate wing translation and rotation. Notably, during this process, users view the wings within a display panel, observing the extracted wing boundaries rather than the complete wing images. In this process, one wing remains fixed as a reference, while the position of the other wing can be adjusted solely to achieve alignment. It is important to highlight that the software does not limit users to relying exclusively on visual superimposition. A text box integrated into the software displays distances between outline points, cells' centroids, and junctions. This feature empowers users to quantitatively assess the impact of positional adjustments beyond just visual inspection. As a result, users can determine whether changing the wing's position effectively minimizes these measured distances.

### Automated Superimposing

The automated superimposition is often more efficient and time-saving than the manual superimposition. The *Auto superimposing* button in Figure S18 is embedded for this purpose. This approach utilizes Particle Swarm Optimization (PSO) to perform translations and rotations on one wing, minimizing the differences between wing outlines [171]. The PSO algorithm works with a population called a swarm consisting of candidate solutions known as particles. The PSO algorithm is configured with the following parameters:

- Number of variables: 3 (x, y, theta)
- Maximum iterations: 40
- Number of particles: 30
- Inertia weight (w): 1
- Damping coefficient (wDamp): 0.95
- Cognitive parameter (C1): 1.2
- Social parameter (C2): 1.2

Figure 6.2a illustrates the extracted outlines of the right and left wings. Before initiating the PSO optimization, the centroids of the wings are aligned (Figure 6.2b). Subsequently, the method assesses the distances between corresponding points along the wing outlines (Figure 6.2c). The PSO's variables consist of x, y, and theta. 'x' and 'y' correspond to translation along the x and y axes, while 'theta' represents the rotation angle of the wing (Figure 6.2d). In each iteration of the optimization process, the particle with the minimal distance is designated as the best particle. After iterative experimentation, it has been determined that the PSO converges to the optimal superimposition after 40 iterations (Figure 6.2e). The procedure results in the alignment of the two wings. Subsequently, the method identifies corresponding cells on each wing by calculating the minimum distance between cells, facilitating the quantification of wing asymmetry (Figure 6.2f). Code S11, developed for automated superimposition, is detailed in the Appendices.

## 6.2.4 Asymmetry Metrics

Once the wings' boundaries are extracted, the methodology proceeds to quantify the area, length, width, and circularity of the wing cells and the entire wing. The software superimposes wings by employing the PSO technique, thereby establishing correspondence between cells on both wings. The WingAnalogy feature, integral to the software, evaluates wing asymmetry based on several criteria:

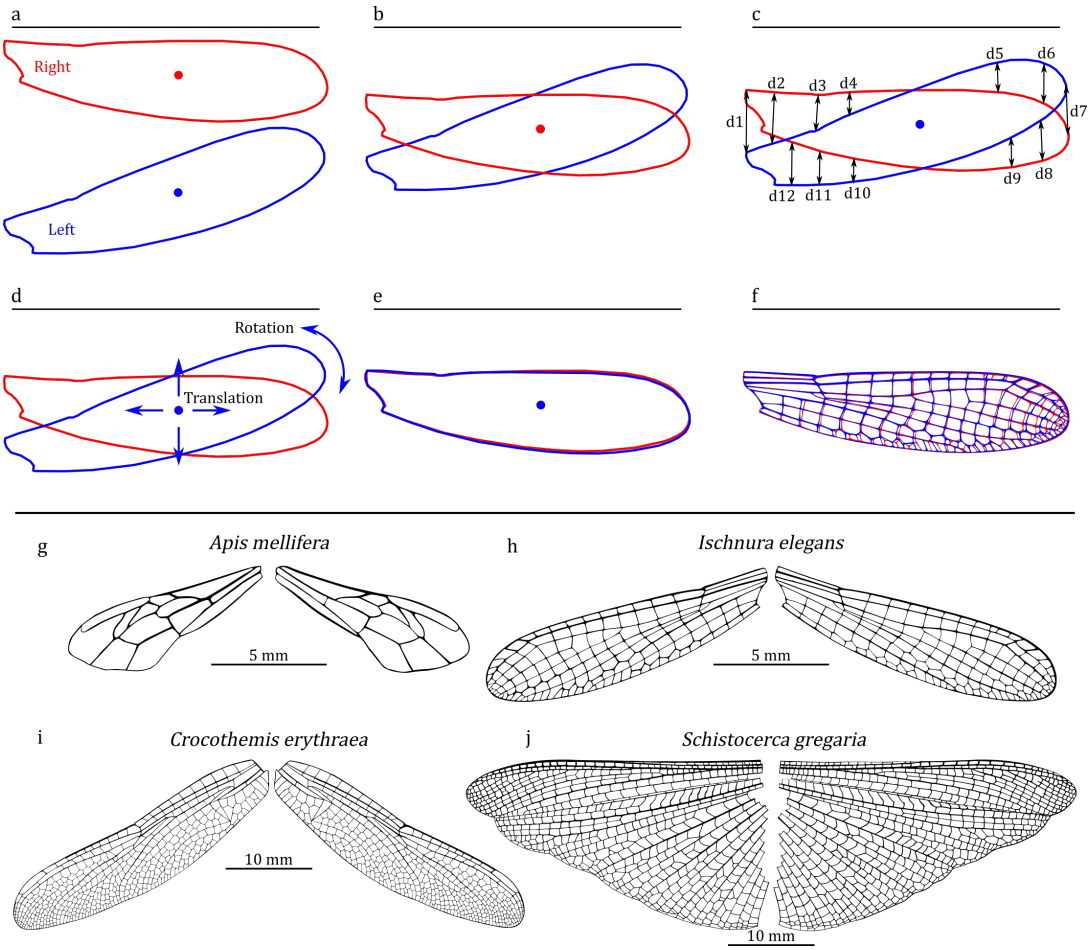



Figure 6.2: Wing superimposition process and case study selection. (a) The outlines of the right and left wings. (b) The outlines after aligning them at a common center. (c) The measurement of distances between two wing outlines. (d) The translation and rotation of one wing for alignment. (e) The outlines of two wings after successful alignment. (f) Both wings, including their cells, after superimposition. Case study pairs include (g) forewings of *Apis mellifera* (Honeybee, Hymenoptera), (h) forewings of *Ischnura elegans* (Blue-tailed damselfly, Odonata: Zygoptera), (i) forewings of *Crocothemis erythraea* (Scarlet dragonfly, Odonata: Anisoptera), and (j) hindwings of *Schistocerca gregaria* (Desert locust, Orthoptera).

1. **Normalized Root Mean Square Error and Regression.** The software computes these metrics for the area, length, width, and circularity of cells corresponding to each other. This computation offers insights into the extent of similarity or divergence between the left and right wings.

2. **Mean Distance Measurements.** Another criterion for assessing asymmetry involves measuring the distances among the wing outlines, junctions, and centroids of cells.

3. **Subtraction of Amounts.** Our methodology quantifies the number of cells and junctions and measures the whole wing's area, length, width, and perimeter. Utilizing the disparity in these counts and amounts between wings as an asymmetry criterion adds another layer of assessment.

This multifaceted approach gives a wide range of metrics and analyses to evaluate wing asymmetry thoroughly. By encompassing individual cell characteristics and overarching wing features, WingAnalogy provides a comprehensive and detailed perspective on the

asymmetry of insect wings. Code S12, developed for computing the asymmetry of two wings, is described in the Appendices.

### 6.2.5 Graphical User Interface

We have developed a user-friendly graphical interface for those who may not be familiar with coding. This interface is designed using MATLAB App Designer (version 2020b) and contains multiple windows, each serving specific functions. Detailed software information is available in the Appendices. Moreover, *WingAnalogy Video S2* to *WingAnalogy Video S12* are available in Supplementary Data, prepared to show how the software works [100].

The software encompasses several windows, with Figure S14 illustrating the main window. In this window, users can create projects from the file and menu (Figure S15). Each project can include multiple pairs of wings, which can be added using the 'Add New Pair' button. Clicking this button opens another window, depicted in Figure S16. Here, users can import wing images and access various Figures related to wing image segmentation and geometric features, similar to what's shown in Figure S17. Users can also scale the images (Figure S16), superimpose them (Figure S18), and view comparison results through statistics, histograms with distribution fits, and regression graphs (Figures S19, S20, and S21).

Figure S22 shows a tool embedded in this window that allows users to visualize each wing cell, displaying geometric features such as area, length, width, and circularity. Another tool within this window enables users to divide the wing into a maximum of five sets, each comprising at least three membranes (Figure S23 and S24).

For each pair, users can generate reports regarding the imported wings and the results of their comparison in various formats, including PDF, CSV, or TXT files. After defining each pair, it is added to the list of pairs in the main window (Figure S25). After adding several pairs, users can use the 'Result' menu from the main window to access another window, as illustrated in Figure S26, to compare all added pairs. In this new window, several comparisons between pairs, such as NRMSE, regression of area, length, width, and circularity, differences in wing areas, and the subtraction of the number of cells and junctions in wing pairs, are available. The EXE file of WingAnalogy can be accessed from the Supplementary Data [100].

### 6.2.6 Validation, Testing, and Performance Evaluation

WingAnalogy is a multifunctional software featuring numerous windows and functions that seamlessly interact with one another. Several approaches were conducted to assess the software performance, encompassing input and output validation for all functions and evaluating their interactions within the software. WingAnalogy was meticulously designed to be adaptable, accommodating wings from various insect orders, each presenting distinct geometric characteristics and complexities.

One of the fundamental functions within WingAnalogy is insect wing image segmentation, which involves extracting the wing boundary and its constituent cells while measuring their area, length, width, circularity, and the locations of junctions. Due to its pivotal role, this function required thorough validation. Additionally, another critical function of WingAnalogy involves quantifying the asymmetry of wing pairs. In this study, we utilized an image of *Apis mellifera* (honeybee, Hymenoptera) from Figure 6.2g to validate both of these functions.

To achieve this, we measured the area, length, and circularity of all cells within the right and left wings of the Honeybee using ImageJ [61]. We then employed our

segmentation method within WingAnalogy to measure the same features. Subsequently, we assessed the accuracy of our approach to evaluate its reliability. We utilized the equation 6.2 to compute the accuracy percentage.

$$Accuracy = 100 * \frac{|Result_{ImageJ} - Result_{WingAnalogy}|}{Result_{ImageJ}} \tag{6.2}$$

Furthermore, we leveraged the results obtained from ImageJ to compute the regression and NRMSE for corresponding cells between the left and right wings, enabling us to quantify the asymmetry of the honeybee wings. We then compared these results to those obtained through WingAnalogy. Moreover, wings from other insect species, including *Ischnura elegans* (Blue-tailed damselfly, Odonata: Zygoptera)(Figure 6.2h), *Crocothemis erythraea* (Scarlet dragonfly, Odonata: Anisoptera)(Figure 6.2i), and *Schistocerca gregaria* (Desert locust, Orthoptera)(Figure 6.2j) are selected to evaluate the performance of the software with more complex wings. The wing images in Figure 6.2g-j were acquired using a high-resolution scanner (Epson Perfection V850 Pro). Before their utilization in the software, all images underwent editing.

## 6.3 Result

The primary focus of this study revolves around the outcomes generated by WingAnalogy software when applied to pairs of insect wing images. In Figure 6.2 g-j, we have showcased four pairs of wings that serve as our test cases for assessing the software performance in producing results.

Once a new pair is defined and two wing images are added, the software initiates an automated process. It begins by segmenting the wing image, extracting cell boundaries, defining the wing outline, and identifying the locations of junctions. Subsequently, it quantifies parameters, such as the area, length, width, and perimeter of the wing outline and the area, length, width, and circularity of wing cells. Figure 6.3a-d shows the contour related to wing cell area distribution, length, width, and circularity generated by WingAnalogy. Beneath each contour, the histogram plot displays the differences in the areas, lengths, widths, and circularities of the left and right wing cells. Next, users can scale the wing images to their actual size and superimpose them for further analysis. Figure 6.5m-p illustrates the superimposed wings. Additionally, the software allows users to partition the wing into a maximum of five cell sets, as shown in Figures 6.5a,d,g, and j. Once these steps are completed, users can generate a comprehensive report for that particular wing pair in three different formats: PDF, TXT, and CSV.

The PDF file contains six chapters, each serving a distinct purpose.

1. **Image Specifications (Chapter 1 and 2).** These chapters contain information related to the first and second wing image specifications. Details, such as the wing images themselves, cell and junction counts, wing area, perimeter, length, and width, and comprehensive data on cell area, length, width, and circularity, are presented. Contour distributions and histograms illustrating cell characteristics can also be found in these chapters.

2. **Comparison Results (Chapter 3 and 4).** Chapters 3 and 4 provide insights into the results of comparing wings. Chapter 3 pertains to comparisons with the first image as the reference, while Chapter 4 uses the second image as the reference. These chapters include information on metrics such as NRMSE, regressions, distances between cell centroids, and junctions.

3. **Results for Wing Sets (Chapters 5 and 6).** The final chapters focus on results related to cell sets of the wings. This includes details on the cell sets' area and regression and NRMSE values based on cell sets' area, length, width, and circularity.

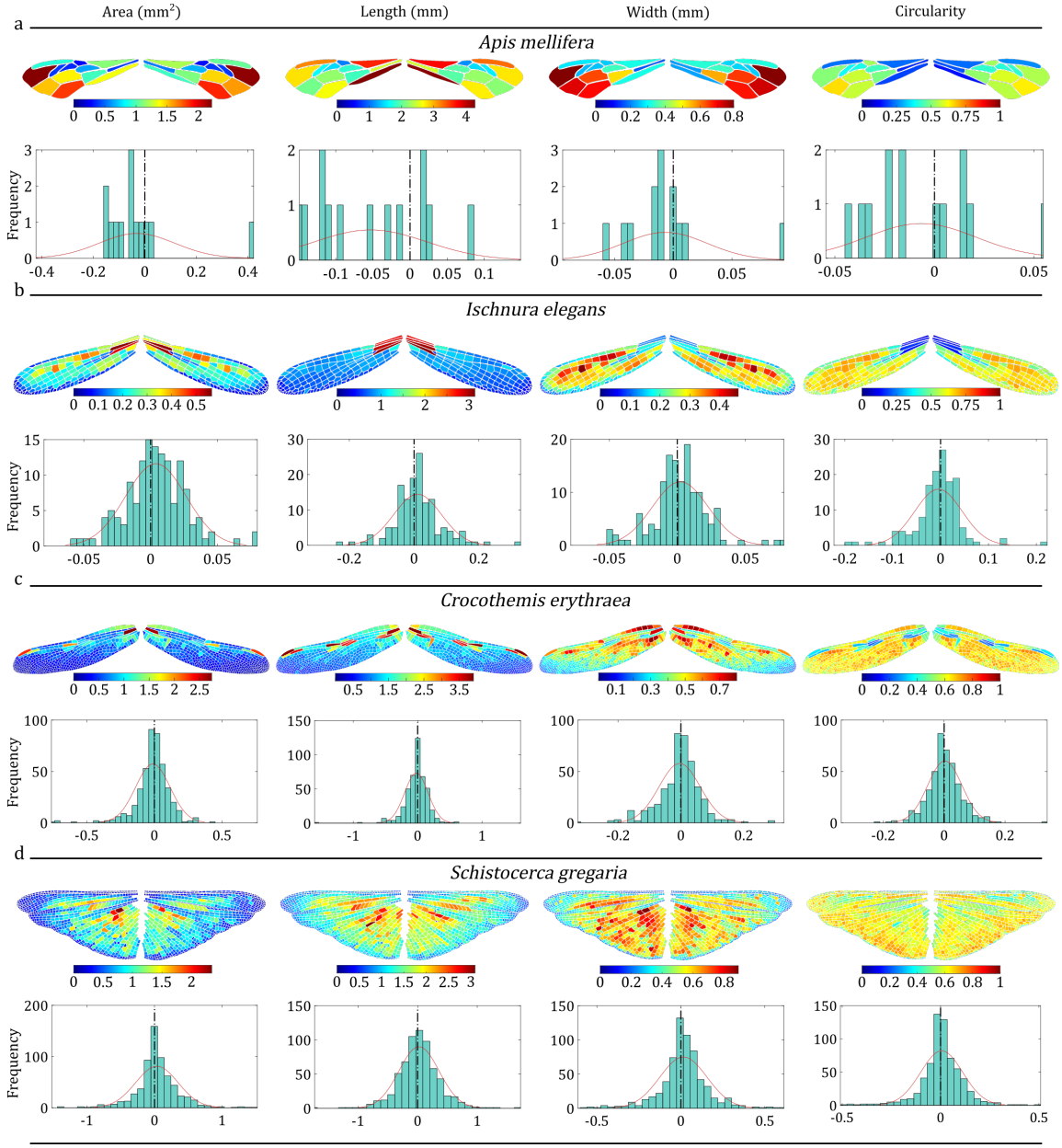



Figure 6.3: Filled contour plots depicting the distribution of wing cell area, length, width, and circularity, along with corresponding histograms showing a distribution fit for (a) *Apis mellifera* (Honeybee, Hymenoptera), (b) *Ischnura elegans* (Blue-tailed damselfly, Odonata: Zygoptera), (c) *Crocothemis erythraea* (Scarlet dragonfly, Odonata: Anisoptera), and (d) *Schistocerca gregaria* (Desert locust, Orthoptera).

Users can access a comprehensive compilation of all quantities and values in the TXT and CSV files by comparing wing pairs and image specifications. These files serve as a detailed data repository for further analysis and record-keeping. All PDF, TXT, and CSV files regarding the test cases in this study are available in the Supplementary Data [100].

The primary objective of WingAnalogy is to quantify wing asymmetry, which is achieved by extracting geometric features and subsequently computing the results based on predefined asymmetry metrics. This functionality enables the comparison of individual pairs and facilitates the comparison of multiple pairs simultaneously.

Figure 6.4 illustrates the outcomes of asymmetry measurements generated by WingAnalogy for all four pairs. Figures 6.4a and c depict each pair regression and NRMSE values, considering parameters, such as area, length, width, and circularity of wing cells. In contrast, box plots in Figures 6.4b and d represent the regression and NRMSE values for cell-specific characteristics (area, length, width, and circularity) across all pairs.

Figure 6.4e provides insights into the mean distance between cell centroids, wing junctions, and wing outlines, serving as an additional metric for assessing wing asymmetry. The boxplot in Figure 6.4f visually presents these distances for all pairs. Figures 6.4g and h mirror the format of Figures 6.4e and f, but they pertain to measuring the standard deviation of mean distances.

Furthermore, Figure 6.4i offers a visualization of the subtracted values representing the differences in the number of cells and junctions among all pairs. The corresponding boxplot in Figure 6.4j displays these values collectively for all pairs. Finally, Figures 6.4k and l follow a similar pattern to Figure 6.4i and j, focusing on differences in the total wing area among the pairs. The Appendices documents the data corresponding to Figure 6.4 in Tables S1 and S2.

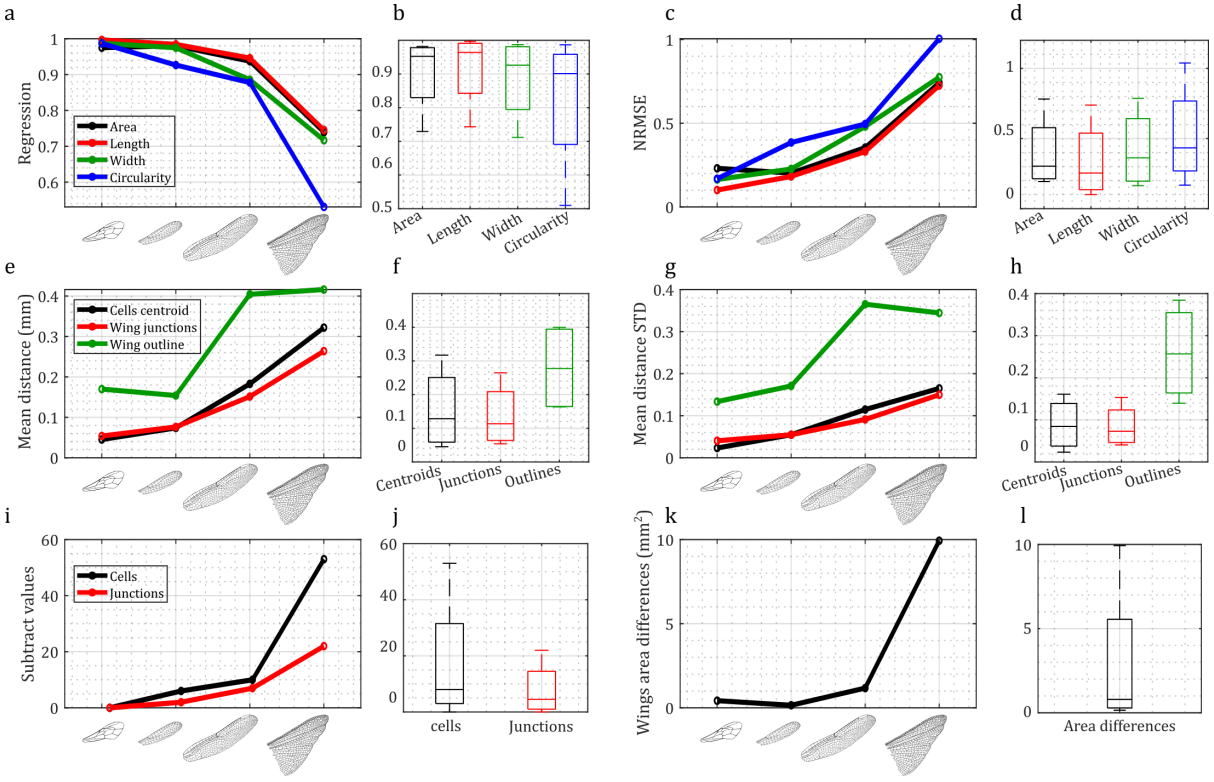



Figure 6.4: Wing asymmetry comparison for the four case studies of wing pairs. (a),(b) Regressions for area, length, width, and circularity across four wing pairs, and respective boxplots. (c),(d) NRMSE for cell area, length, width, and circularity, and respective boxplots. (e),(f) Mean distances between cell centroids, wing junctions, and wing outlines, and respective boxplots. (g),(h) Standard deviation of mean distances between cell centroids, wing junctions, and wing outlines, and respective boxplots. (i),(j) The subtracted values in the number of cells and junctions and respective boxplots. (k),(l) Wing area and respective boxplot.

Additional results are related to the sets of cells depicted within the wings. Figures

6.5a,d,g, and j depict the wings of the honeybee, damselfly, dragonfly, and locust, each divided into five cell sets distinguished by different colors. Next to each wing, the outcomes of set-to-set comparisons are displayed, including regressions and NRMSE values for cells associated with each set (Figures 6.5b,c,e,f,h,i,k, and l).

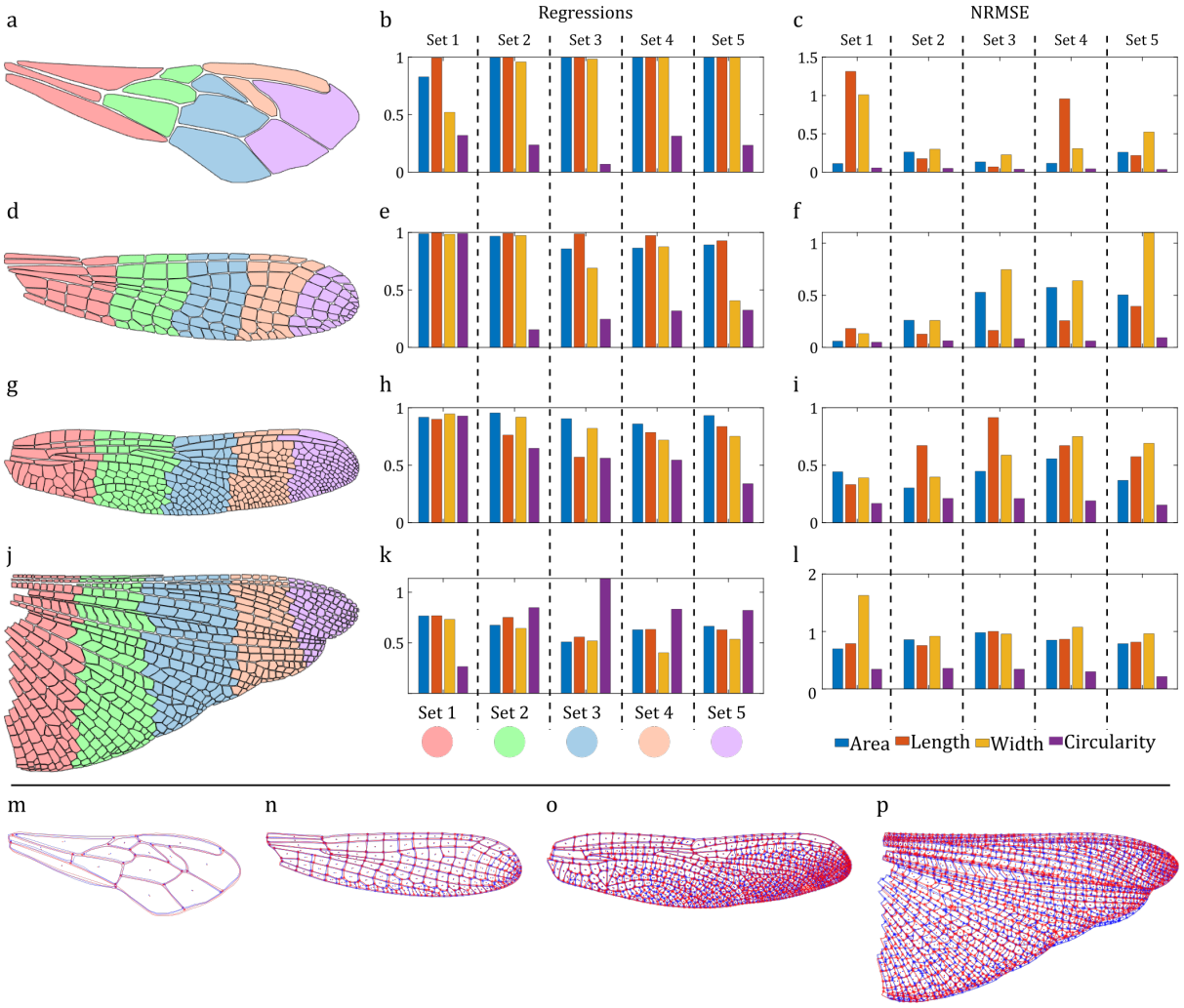



Figure 6.5: Analysis of wing cell sets and Superimposing Results. (a, d, g, j) Display the defined wing cell sets for honeybee, damselfly, dragonfly, and desert locust. (b, e, h, k) Depict the regressions of cell area, length, width, and circularity, for the respective insect wing pairs in the case studies. (c, f, i, l) Showcase the results of the Normalized Root Mean Square Error (NRMSE) calculations for cell area, length, width, and circularity in the four insect wings studied. (m, n, o, p) Illustrate the superimposed wings from each of the four case studies, providing insights into the alignment and comparison of these wing pairs.

In WingAnalogy, the software capabilities extend to comparing different pairs based on their wing cell sets, as shown in Figure 6.6. This Figure offers a comprehensive view of the analytical insights derived from these comparative analyses. Figures 6.6a-e present detailed results for the first to fifth wing cell sets across all pairs. Within each panel, multiple plots provide distinct insights. Linear plots in these panels correspond to essential metrics, including Regression, NRMSE, Mean Distances of Cell Centroids, and their Standard Deviations. Accompanying each linear plot, a boxplot summarizes the results across all pairs.

Figures 6.6f-o complement the initial panels with a series of boxplots, each assessing various wing cell sets based on specific metrics. Figures 6.6f-i showcase boxplots depicting

regression values for cell area, length, width, and circularity. Figures 6.6j-m presents boxplots summarizing NRMSE value for cell area, length, width, and circularity. Figure 5o provides boxplots depicting the mean distances of cell centroids, offering insights into wing cell set comparisons. Figure 6.6p illustrates boxplots representing standard deviations of mean distances, further enriching set comparisons. Figure 6.6 provides a comprehensive and structured overview of the results when comparing pairs based on their cell sets within the WingAnalogy software. All the details of Figure 6.6 are meticulously documented within Table S3 in the Appendices.

### 6.3.1 Validation

The assessment of WingAnalogy performance was carried out to validate its outcomes and assess its reliability. To achieve this, we employed ImageJ, a renowned tool recognized for its accuracy in image analysis. Our focal point for this validation procedure was the honeybee wing. Consequently, we quantified the area, length, and circularity of all cells within the left and right wings using our developed method and subsequently with ImageJ. The resulting right and left-wing data are stored in Tables S4 and S5 within the Appendices file.

The findings indicate that measurements of area, length, and circularity for the right wing exhibited merely an average difference of 0.056%, 1.59%, and 3.7%, respectively. These results underscore the high accuracy and precision of our methodology. Similarly, in the case of the left wing, average differences in area, length, and circularity were recorded at 0.096%, 1.6%, and 3.8%, respectively. These outcomes further emphasize the reliability and effectiveness of our approach.

Additionally, we conducted assessments of regression and NRMSE between the cell area, length, and circularity of the right and left wings, utilizing results obtained from ImageJ as the benchmark, and subsequently compared these with the outcomes generated by WingAnalogy. Remarkably, the disparities observed in the regression values for area, length, and circularity were merely 0.01%, 0.04%, and 0.1%, respectively. Moreover, the differences in NRMSE for area, length, and circularity were measured at 0.3%, 3.8%, and 3.6%. These findings further underscore that our methodology is as accurate as ImageJ, reaffirming the robustness of our approach.

## 6.4 Discussion

WingAnalogy is developed to streamline the process of analyzing insect wing geometry and quantifying wing asymmetry. This approach deviates from traditional landmark-based methods, which grapple with inherent limitations. The constraints of landmarking necessitate the precise identification of identical landmarks across all wings, a tough challenge that specific landmarks may be absent due to factors like wing mutations, rendering such wings unsuitable for analysis—despite their potential richness in characters [50, 51]. Furthermore, landmark-based approaches encounter difficulties when comparing wings from distinct insect orders, as demonstrated in our case studies. These methods typically focus on a handful of landmarks while overlooking numerous other critical aspects of wing morphology. Additionally, landmark determination often relies on manual input, a time-intensive process that poses challenges for field studies [52–54]. In contrast, our methodology adopts a multifaceted approach, considering parameters, such as cell area, length, width, circularity, junction locations, cell and junction counts, inter-cell centroid distances, and wing outline attributes (area, length, width, perimeter). This breadth of

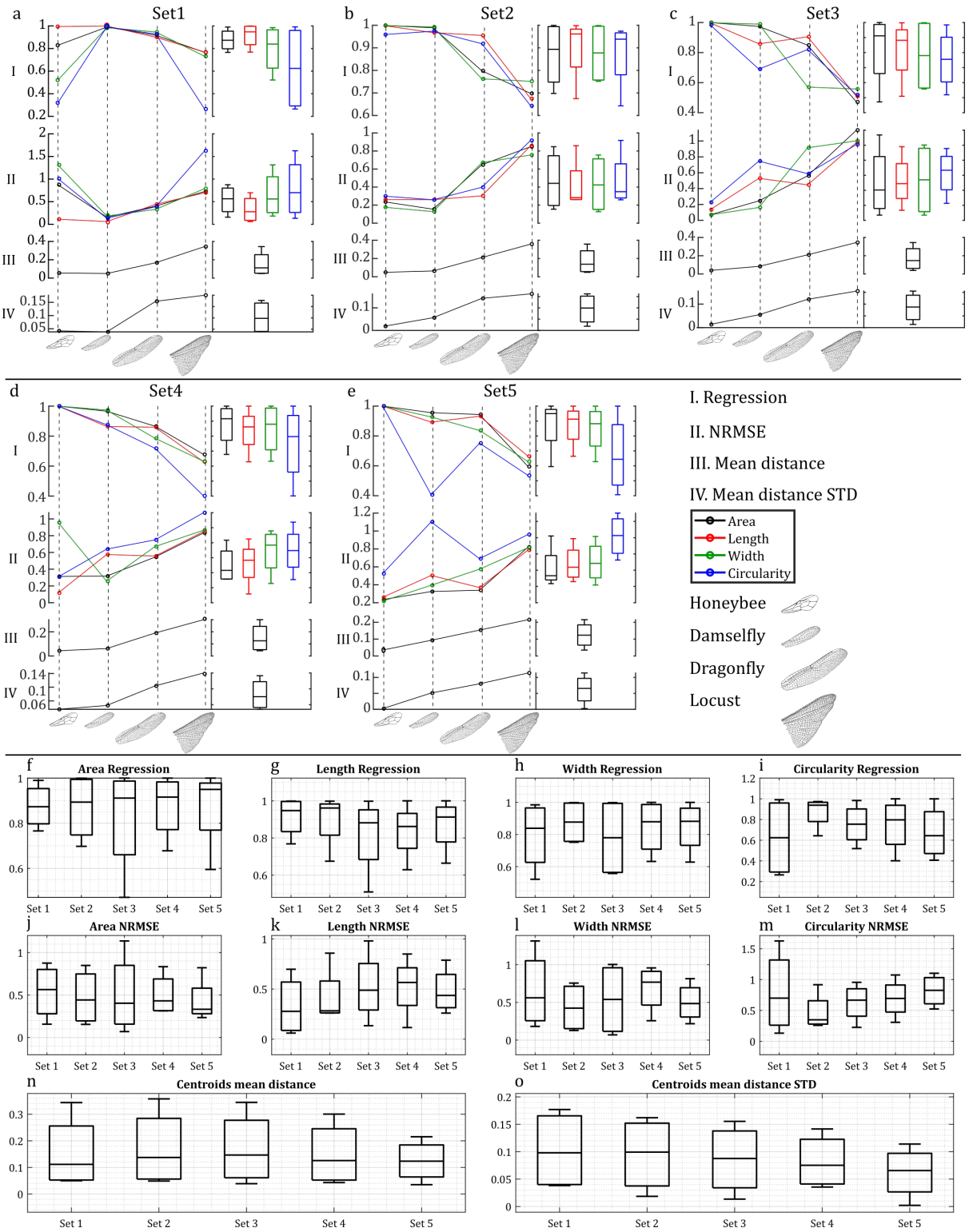



Figure 6.6: Analysis of defined wing cell sets for the honeybee, damselfly, dragonfly, and desert locust wing pairs. (a-e) The outcomes of the Regression, Normalized Root Mean Square Error (NRMSE), mean distances of cell centroids, and the standard deviation of mean distances of cell centroids for Sets 1 to 5 across all pairs. (f-i) Feature boxplots corresponding to the regression of wing cell sets' area, length, width, and circularity for cell sets 1 to 5. (j-m) NRMSE values for the area, length, width, and circularity of cells within Sets 1 to 5 for all pairs. (n,o) The mean distance of wing cell centroids and their standard deviations for wing cell sets 1 to 5 across wing pairs, respectively.

parameters affords researchers a rich array of analytical options. Moreover, the automated superimposition feature enhances the examination of wing asymmetry. Nevertheless, the process of preparing images for WingAnalogy still involves manual cleaning, which can be time-consuming, especially when dealing with a substantial number of images. We are actively working on enhancing the software to minimize the effort required to prepare images for its use.

While the primary objective of this study is to underscore the eligibility of WingAnalogy, a brief discussion of the results concerning the asymmetry comparison of four case study wing pairs can be illuminating. As our results in Figure 6.4 demonstrate, wing asymmetry tends to increase with wing complexity. Notably, the locust hindwings exhibit the highest degree of asymmetry among the case study pairs, whereas honeybee forewings show greater symmetry. Examining the division of wings into five distinct cell sets reveals varying degrees of asymmetry across these cell sets. Of particular interest is the dragonfly wing, which indicates that the base exhibits the highest degree of similarity, with asymmetry increasing along the wing, particularly in terms of circularity—a noteworthy observation signifying that while wing size may not significantly differ, cell shape, as indicated by circularity, exhibits obvious variations along the wing [167]. It is worth mentioning that this study did not employ equal-area divisions within the wings, which may have influenced the results. In subsequent sections, we delve into a comprehensive discussion of WingAnalogy and its versatile applications across different domains.

### 6.4.1 Advancements in Insect Wing Analysis

In recent years, insect wing analysis has undergone a transformative evolution, primarily driven by the development of sophisticated software tools [55, 56, 59, 60]. These advancements have revolutionized our approach to studying insect wing morphology and symmetry. Automation and efficiency have become paramount as software automates intricate tasks like image segmentation and cell identification, significantly enhancing the accuracy and speed of analysis. High-resolution imaging technology has enabled researchers to capture intricate wing details, allowing for precise measurements of features and fine-scale asymmetry. Moreover, developing standardized quantitative metrics has expanded our understanding of wing morphology. The ability to perform local comparisons by dividing wings into cell sets has uncovered precise insights, while the capacity for large-scale comparisons has facilitated comprehensive studies across diverse insect populations. Visualization options and detailed reporting further enhance transparency and reproducibility. With interdisciplinary applications and emerging technologies, like machine learning, the future of insect wing analysis holds the potential for uncovering new dimensions of understanding in this field.

### 6.4.2 Automation and Efficiency

Incorporating automation and efficiency in insect wing analysis represents a significant advancement in our research capabilities. As advanced software tools such as WingAnalogy have come to the forefront, processes that require a significant amount of time and manual effort, such as image segmentation and cell identification, have been streamlined to near perfection. Researchers no longer deal with manually outlining wing structures; automation ensures consistency and precision in every analysis. This automation not only advanced the analytical process but also minimizes human error, resulting in more reliable and reproducible data [43, 55, 60, 156, 172].

### 6.4.3 Precision and Accuracy

WingAnalogy demonstrates exceptional precision and accuracy in its measurements, as demonstrated by a validation process. To ensure the reliability of its results, a comprehensive validation was undertaken using ImageJ as a benchmark tool. This validation employed the left and right forewings of the honeybee, with individual cell area, length, and circularity measurements conducted independently using both ImageJ and WingAnalogy. Surprisingly, when the results were compared, there was a perfect alignment between the measurements obtained from WingAnalogy and those from ImageJ. This underscores the software's remarkable ability to replicate measurements with exceptional precision consistently. Moreover, the validation extended to encompass NRMSE and Regression analyses of cell data, where the results obtained from ImageJ were compared with those generated by WingAnalogy, once again revealing an impressive alignment. These findings not only witness to the precision and accuracy of WingAnalogy but also underscore its potential as a reliable tool for researchers in insect wing analysis, ensuring the generation of reproducible data.

While the validation confirms the robustness and accuracy of WingAnalogy, it is essential to note that the quality and resolution of input images play a crucial role in the precision of the analysis. High-resolution images are necessary to capture fine details and ensure accurate segmentation. We recommend a minimum resolution of 300 DPI for scanning insect wings to achieve reliable results. For smaller wings, such as those of Drosophila, higher resolutions (e.g., 600 DPI or higher) are advisable. This recommendation ensures that all relevant features are captured accurately, enabling WingAnalogy to perform optimally. By adhering to these guidelines, researchers can mitigate the impact of image quality on the analysis, ensuring that the measurements of fine-scale asymmetry and other morphological features remain precise and reliable.

### 6.4.4 Comprehensive Data Analysis

WingAnalogy offers a comprehensive suite of data analysis tools that collectively provide a comprehensive data analysis of wing asymmetry, providing the diverse needs of researchers in the field. The breadth of data analysis provided by the software encompasses several key solutions for metrics, including regression analysis, NRMSE calculations, centroid distance measurements, and counts of cells and junctions.

- **Regression Analysis.** Regression analysis is a fundamental statistical technique that helps to quantify the relationship between variables. In insect wing analysis, it enables researchers to assess the degree of linear association between corresponding cells in paired wings. By examining the slope of the regression line, users can gain insights into the overall symmetry or asymmetry trends in wing structures. A regression analysis is valuable for identifying systematic deviations from perfect symmetry, providing essential information about the magnitude and direction of asymmetry.

  The regression analysis aims to compare the morphological metrics of corresponding cells between pairs of wings to quantify asymmetry. A robust approach is used, where each image is treated as a reference in turn, ensuring comprehensive evaluation of all cells. For each pair of wing images, the analysis is performed twice: first using Image 1 as the reference and then using Image 2 as the reference. When Image 1 is the reference, each cell in Image 1 is matched to the closest cell in Image 2 based on centroid distance. This method ensures that, even if Image 2 contains

a different number of cells, the cells in Image 1 are adequately matched. In cases where Image 1 has 300 cells and Image 2 has 320 cells, the extra cells in Image 2 are initially ignored. The process is then repeated with Image 2 as the reference to ensure that these additional cells are also considered.

This bidirectional matching process helps mitigate the impact of segmentation errors, such as instances where two cells are treated as one or additional cells are detected due to mutations or wing injuries. By performing the regression analysis on the matched sets of cells and using linear regression to compare the morphological metrics (e.g., area, length, circularity) between the corresponding cells, asymmetry patterns can be directly compared and identified. The regression analysis uses the set of points representing the metrics of matched cells rather than the regression equations themselves. This approach allows for a more direct comparison of asymmetry patterns. The system is designed to be resilient to segmentation differences. By using both images as references in turn, the method ensures that all cells are evaluated, thereby reducing the impact of segmentation errors on the overall analysis. This method also accommodates potential biological variations, such as mutations or wing injuries, by ensuring that all detected cells are included in the analysis, either as primary matches or secondary references.

- **NRMSE (Normalized Root Mean Square Error) Calculations.** NRMSE is a critical metric for evaluating the accuracy of wing superimposition and cell correspondence in WingAnalogy. It quantifies the goodness of fit between corresponding cells in paired wings, measuring the discrepancies between their shapes and sizes. The normalization factor accounts for differences in scale, ensuring that NRMSE considers relative and absolute differences. This metric provides researchers with a quantitative measure of the quality of superimposition and cell matching, aiding in assessing wing asymmetry.
- **Centroid Distance Measurements.** WingAnalogy calculates the distances between centroids of corresponding cells and junctions in paired wings. These centroid distance measurements offer insights into the spatial distribution of asymmetry. This information is valuable for understanding the magnitude of asymmetry and its spatial patterns across the wing surface.
- **Counts of Cells and Junctions.** Quantifying the number of cells and junctions in paired wings is fundamental to understanding their structural differences. By providing counts of these elements, WingAnalogy offers researchers an essential but crucial metric for characterizing wing morphology. Comparing cell and junction counts between left and right wings can reveal differences that may contribute to overall wing asymmetry.

These metrics provided by WingAnalogy offer a multifaceted view of wing asymmetry, encompassing both quantitative and spatial aspects. Researchers can use these metrics to delve deeply into the structural variations within insect wings, gaining a comprehensive understanding of asymmetry patterns. This complete data analysis capability empowers researchers to conduct thorough investigations into the factors contributing to wing asymmetry, advancing our knowledge in entomology, ecology, and evolutionary biology.

### 6.4.5 Wing Cell Sets Analysis

The wing cell sets analysis feature in WingAnalogy is a valuable tool, enabling users to dissect insect wings into distinct sets for localized comparisons. This feature holds great

utility in various research scenarios, offering a perspective on wing morphology within and between species. Researchers can utilize this capability to explore intricate variations in different wing cell sets, showing specific adaptations or ecological implications. For instance, when studying wing variation within a single species, this feature facilitates the examination of region-specific adaptations or asymmetry patterns. Additionally, in comparative studies involving multiple species, the wing cell sets analysis allows for identifying species-specific characteristics in particular wing segments, aiding in discerning evolutionary trends and ecological niches.

### 6.4.6 Multi-Pair Comparison

The capability of WingAnalogy to compare the results of multiple wing pairs simultaneously is a powerful feature that enhances the scope and depth of research in insect wing analysis. By facilitating the simultaneous analysis of numerous wing pairs within a single project, researchers gain the capacity to draw broader and more robust conclusions. This feature is particularly advantageous when investigating wing asymmetry across various specimens or species. Researchers can leverage the collective data from multiple pairs to identify overarching patterns, trends, or variations. Whether looking at intraspecific variation within a species or interspecific differences between multiple species, this capability allows for a comprehensive exploration of asymmetry dynamics. It provides a broader context for understanding the evolution, ecology, and adaptations of insects, enhancing the depth of insights derived from the wing morphology and asymmetry analysis.

### 6.4.7 Report Generation

The generation of detailed reports in PDF, CSV, and TXT formats by WingAnalogy holds significance in insect wing analysis. These reports serve as comprehensive and accessible repositories of research findings, streamlining the presentation of results and enhancing the efficiency of data sharing and collaboration among researchers. In PDF format, the reports offer a visual and standardized representation of asymmetry metrics, aiding in clearly and concisely communicating research outcomes. The inclusion of CSV and TXT formats caters to the analytical needs of researchers, allowing for further data processing and statistical analyses. Moreover, these report formats promote transparency and reproducibility by providing a structured and documented account of the analysis process.

### 6.4.8 User-Friendly Interface

WingAnalogy has a user-friendly graphical interface that empowers researchers, regardless of their technical background, to easily access the full potential of the software. Its user-friendly design guides users through each step of the wing analysis process, from project creation and image import to result visualization and report generation. The software offers clear and straightforward options for setting parameters, scaling wing images, and defining wing cell sets of interest, ensuring that users can efficiently tailor their analyses to specific research objectives.

Moreover, the interface provides real-time feedback and visualizations, allowing researchers to preview and validate their selections and configurations, enhancing the accuracy of their analyses. Comprehensive documentation and tutorials further complement the interface, ensuring that users can quickly become proficient in utilizing WingAnalogy for their research needs. The user-friendly interface lowers the barriers for researchers

who are not experts in the wing analysis and streamlines the workflow for experienced users, saving time and effort.

## 6.5 Applications, Current Limitations and Future Research

Indeed, discussing the applications and potential future research avenues for WingAnalogy is essential to showcase its versatility and its broader impact on various research domains.

### 6.5.1 Applications

1. **Taxonomy and Species Identification.** Wing morphology plays a crucial role in species identification and taxonomy [173–175]. WingAnalogy can be a valuable tool for entomologists and taxonomists, enabling them to accurately quantify wing asymmetry and variations within and between species. This can aid in refining species descriptions and understanding evolutionary relationships.

2. **Ecology and Adaptation.** Understanding how wing morphology relates to ecological niches and adaptations is vital. WingAnalogy can help researchers explore how environmental factors influence wing asymmetry and morphology, revealing an understanding of species' ecological roles and transformations to different habitats [176].

3. **Evolutionary Biology.** WingAnalogy can contribute significantly to the study of evolutionary biology. Researchers can use it to investigate patterns of wing structure and asymmetry in fossils, allowing for insights into the evolution of insect wings over geological time scales [177, 178].

4. **Biomimetics and Engineering.** In biomimetics and engineering, insights from insect wing analysis can inspire the development of innovative technologies, such as improved aerodynamics, robotics, and drones. WingAnalogy can aid in precisely characterizing natural wing structures for subsequent bioinspired design [21, 133, 179].

### 6.5.2 Current Limitations

In the current version of WingAnalogy, the length of cells is defined as the maximum distance between the boundaries of each cell. This definition is straightforward and provides a consistent measure of the longest dimension of the cell. However, we acknowledge that this approach might define the length as the diagonal in rectangular cells and that the width, calculated as area/length, can be a poor approximation. To address this, we plan to enhance WingAnalogy in future versions by implementing ellipse fitting for each cell. This will allow us to determine the major and minor axes, providing more accurate measures of length and width. The ellipse-fitting approach will offer a more refined and meaningful representation of cell dimensions, especially for cells with irregular shapes. Additionally, while circularity is a useful metric, we recognize that it does not provide information about cell anisotropy. Circularity is limited in its ability to describe the elongation or directional properties of cells. To overcome this limitation, we will include metrics that quantify cell anisotropy using an elongation tensor derived from the cell

contour. This tensor will provide a norm that quantifies cell anisotropy (aspect ratio), allowing for a more comprehensive analysis of cell shape, independent of wing orientation. Furthermore, we acknowledge that the current version of WingAnalogy has constraints in image preparation. The software requires high-quality, clean images to function correctly. It does not work well with wings that have transparent veins or dark regions where the differences between membranes and veins are not recognizable. Ensuring that the images are of high resolution and clear distinction between different parts of the wing is essential for accurate analysis. Moreover, WingAnalogy does not include built-in tools for testing the significance of statistical data. However, it generates comprehensive output data that can be easily exported. For statistical significance testing, users can utilize more dedicated software packages such as SigmaPlot, R, or SPSS. However, this will be a future update to the software. These enhancements will improve the ability of WingAnalogy to identify sources of left-right asymmetry more effectively, providing researchers with detailed and orientation-independent metrics for size and shape. Future versions will also aim to mitigate current image preparation constraints by incorporating more advanced image processing techniques to handle a wider variety of wing images.

### 6.5.3 Future Research Avenues

1. **Morphological Evolution.** Future research could explore how particular wing morphology has evolved within and across insect taxa. This could involve analyzing fossil wings and using WingAnalogy to compare them with extant species, providing insights into long-term morphological trends.

2. **Eco-morphology.** Exploring the connections between wing structure, ecological habitats, and behavior can deepen our understanding of insect ecology. WingAnalogy can be employed to examine how wing characteristics are linked to specific ecological niches.

3. **Climate Change Impacts.** With climate change affecting insect populations, researchers can use WingAnalogy to assess how changing environmental conditions influence wing morphology. This can provide early warnings of ecological disruptions.

4. **Intraspecific Variation.** Studying intraspecific variation in wing morphology using WingAnalogy can reveal valuable information about local adaptations and population dynamics. This could be particularly relevant in understanding how insects respond to environmental gradients.

5. **Biodiversity Conservation.** The software's applicability extends to biodiversity conservation efforts, where researchers can leverage it to examine wing asymmetry in threatened or endangered species. This utilization assists in enhancing conservation strategies and monitoring efforts. More specifically, asymmetry can serve as an informative indicator of population health.

6. **Machine Learning Integration.** Future developments might include integrating machine learning techniques for automated species identification based on wing morphology, further streamlining taxonomy and fieldwork.

# Chapter 7

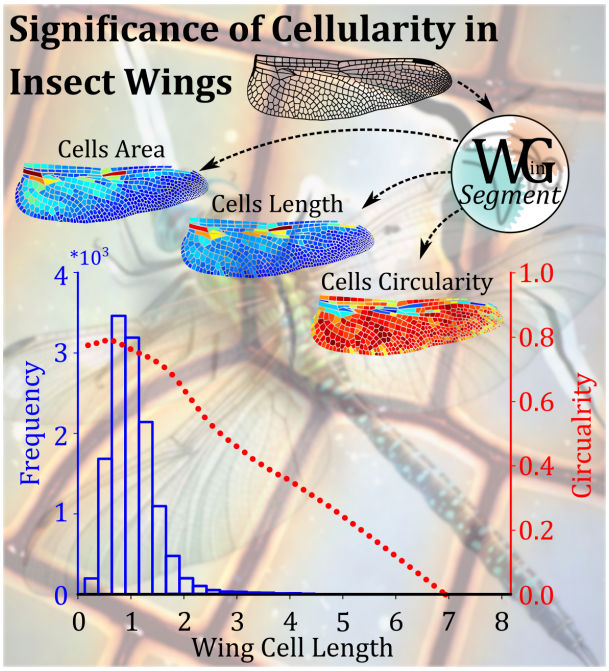
# *Paper V.* Allometric Scaling Reveals Evolutionary Constraint on Odonata wing Cellularity via Critical Crack Length[1]


## Summary

Scaling in insect wings is a complex phenomenon that seems pivotal in maintaining wing functionality. In this study, we investigate the relationship between wing size and the size, location, and shape of wing cells in dragonflies and damselflies (Odonata), aiming to address the question of how these factors are interconnected. To this end, we use WingGram, our recently developed computer-vision-based software, to extract the geometric features of wing cells of 389 dragonflies and damselfly wings from 197 species and 16 families. We found that the cell length of the wings does not depend on the wing size. Despite the wide variation in wing length (8.42 mm to 56.5 mm) and cell length (0.1 mm to 8.5 mm), over 80% of the cells had a length ranging from 0.5 mm to 1.5 mm, which was previously identified as the critical crack length of the membrane of locust wings. We also observed an isometric scaling of cells with maximum size in each wing, which increased as the size increased. Smaller cells tended to be more circular than larger cells. Our results have implications for bio-mimetics, inspiring new materials and designs for artificial wings with potential applications in aerospace engineering and robotics.



[1]This Chapter has been published in *Advanced Science (Wiley)* on April 13, 2024. Reproduced with permission from Wiley.
Citation: S Eshghi, H Rajabi, F Nabati, S Shafaghi, S Nazerian, A Darvizeh, and SN Gorb. Allometric scaling reveals evolutionary constraint on Odonata wing cellularity via critical crack length. *Advanced Science*, 11(23):2400844, 2024. https://doi.org/10.1002/advs.202400844.




**Short Summary:** This study utilizes WingSegment, a Computer-Vision tool, to analyze wing and cell size relationships in 389 Odonata wings. Despite varying wing sizes, statistical analysis shows over 80% of the cells measure between 0.5 and 1.5 mm, implying a link to the crack-stopping function in insect wings. Furthermore, an inverse correlation between cell size and circularity suggests smaller cells are more circular.

## 7.1 Introduction

Insect wings are lightweight segmented structures that balance flexibility and stiffness, achieved through an intricate network of veins interacting with other wing components. This balance is required to generate enough lift for insect flight and keep its structural durability even during collisions [43, 180–184]. One of the fascinating aspects of insect wings is their scaling characteristics, which can vary depending on the species, size, shape, and ecology [185, 186]. Scaling in insect wings can refer to the relationship between wing size and other parts of the insect body or between the wing size and the size of membranes stretching between veins [140]. The veins of an insect's wings form a complex network of interconnected structures that provide support and rigidity to the wing membrane. The interplay between these two components is complex and adaptable, allowing insects to fly and navigate their environments with remarkable efficiency and agility. However, the relationship between wing size and membrane size is still unknown [43, 108, 187].

The wings of the representatives of the order Odonata, including dragonflies and damselflies, have evolved to meet the demands of their flight and maneuverability [179, 188]. The wings of these insects are composed of a complex array of cells, veins, and membranes arranged in a hierarchical structure. The structural characteristics of insect wings are influenced by the size and shape of the wing cells, as well as the overall size of the wing [8, 34, 104, 189, 190]. Understanding the scaling relationships between these parameters can provide insights into the mechanics of the wing and potentially inspire the development of bio-inspired materials and systems [21, 25, 26, 133, 191].

Previous studies have suggested that different Odonata species show different wing length allometries, meaning that the wing size changes at a different rate than the overall body size [192–194]. Other studies also suggest that the size of cells in the locust wing relates to the critical crack length, and the cellular structure of the insect wing is a crack resistance strategy to increase durability [2, 8, 34]. The critical crack length denotes the dimension of a crack in a material where the material undergoes catastrophic

failure or fracture. In simpler terms, it's the point where the combination of stress and crack size becomes so important that the crack quickly spreads and the material fails. However, whether the scaling in wing cells is allometric or isometric remains unresolved. In this study, we address this question by examining the relationship between wing cell geometry relative to the wing by analyzing 389 wings from 197 species of Odonata using WingGram [60] and WingSegment [77]- our previously developed software packages for automated analysis of insect wings [58, 60]. These tools are designed based on computer vision methodologies, such as region growing [71, 72], and line simplification [83] for extracting geometric features of insect wing image. The wings were selected from seven families of damselflies, including Coenagrionidae, Lestidae, Megapodagrionidae, Perilestidae, Platystictidae, Protoneuridae, and Synlestidae, and nine families of dragonflies, including Aeshnidae, Austropetaliidae, Corduligasteridae, Corduliidae, Gomphidae, Libellulidae, Macromiidae, Neopetaliidae, and Petaluridae [55].

## 7.2 Material and Methods

All 389 wing images used in the study were obtained from the publication by Hoffmann et al. [55]. We edited all images to ensure that no noise would influence our measurements. We utilized a Matlab toolbox called WingSegment [77], an improved version of our previously developed toolbox, WingGram [60]. WingSegment employs computer vision and mathematical techniques, including region growing [71, 72], thinning [123], and line simplification [83] algorithms, to extract the geometric characteristics of cells [58–60]. It requires an image of the wing as the input.

Region growing is a method used to identify a domain surrounded by black pixels. This approach considers a randomly selected white pixel within the domain of the initial seed. The process begins by examining the neighboring pixels of this seed. If neighboring pixels are white, their color is changed to grey, and they are treated as new initial pixels. Subsequently, these newly identified white pixels, in the next iteration, examine their neighbors to identify additional white or black pixels. If a pixel is black, it is considered the boundary of the domain. This recursive process continues until all white pixels within the domain are examined. Consequently, all black pixels located on the domain of that region are detected. This process is applied to all cells of the wing automatically until all cells are identified. Once the boundary of each region is detected, the maximum distance between two points inside the region is considered the length of that cell. The number of detected white pixels within the region represents the area of the region. To calculate circularity, Equation 7.1 is employed:

$$C = (4\pi * area)/(perimeter^2) \tag{7.1}$$

Where *area* represents the area of the domain, and *perimeter* denotes the perimeter of the domain, measured from the information extracted from the wing images using the region growing method. Furthermore, the distance between each domain's centroid and the wing's outer line is measured as the distance between cells and margins. All information on the wings examined in this study, including the distribution contour and histogram of wing cell area, length, and circularity, as well as the Excel files containing the measured values, is documented in the supplementary information [195]. Figure 7.1 shows an example of what WingSegment generates from the image of *Acanthagrion chararum* (Coenagrionidae) (Figure 7.1a) and *Aeshna juncea* (Aeshnidae) (Figure 7.1b).

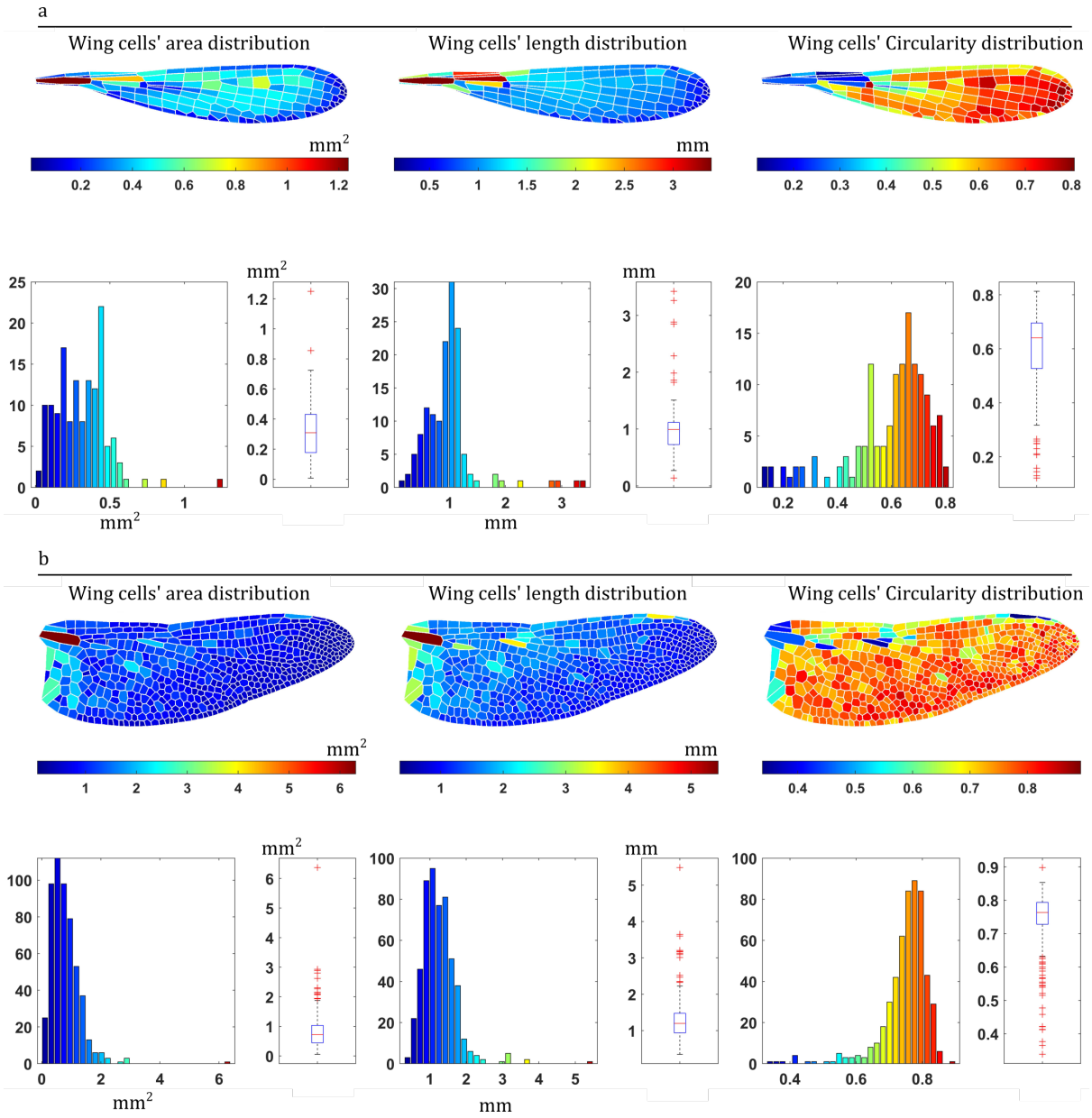



Figure 7.1: Using WingSegment for extracting the geometric features of wing cells in a) right hindwing of *Acanthagrion chararum* (Coenagrionidae) with 18.8 mm wing length, and b) right hindwing of *Aeshna juncea* (Aeshnidae) with 40 mm of wing length. For each wing, the contour distribution, histogram, and box plot representing wing cells' area, length, and width are depicted.

## 7.3 Results

In this study, we measured the area, length, and circularity of wing cells from both dragonflies and damselflies, which were selected from various families. We also measured the area of the wings and the distance of cells from wing margins. All data is documented and is available via the Zenodo repository(See supplementary information [195]), including the family, genus, species, wing position, and length of wings, as well as wing images and the contour of cells' length, area, and circularity.

Figure 7.2 shows the cell length distribution based on the wing length for dragonflies (a) and damselflies (b). Although the wing length varied widely from 10.5 mm to 56.5 mm in dragonflies and 8.42 mm to 37.80 mm in damselflies, the cell length in both dragonflies

and damselflies varied only slightly, ranging from about 0.1 mm to 8mm. However, we found that most cells have a length between 0.5 mm and 1.5 mm, with only a small percentage being less than 0.5 mm or more than 1.5 mm. Specifically, 83% and 82% of cells had a 0.5-1.5 mm length in dragonflies and damselflies, respectively. The additional information presented in Figure 7.3 provides valuable insights into the shape and position of cells in both dragonflies and damselflies. Moreover, Table 7.1 represents the results of the statistical analysis between the wing, and its cells in dragonflies and damselflies.

## 7.4 Discussion

Our analysis indicates that the number of cells increases as the wing area increases (Figure 7.3a.1, b.1). The correlation of 0.77 and 0.89 between the wing area and the number of cells in dragonflies and damselflies supports this conclusion (Table 7.1). Interestingly, as shown in Figure 6.3, we observed that regardless of the area of the wings, the minimum cell length is similar across all wings. Besides, the correlation between the wing area and the minimum size of cell length, as stored in Table 7.1 in all wings in dragonflies and damselflies, is 0.56 and 0.18, respectively. This suggests, especially in damselflies, that there is almost no correlation between the minimum size of the cells and the size of the

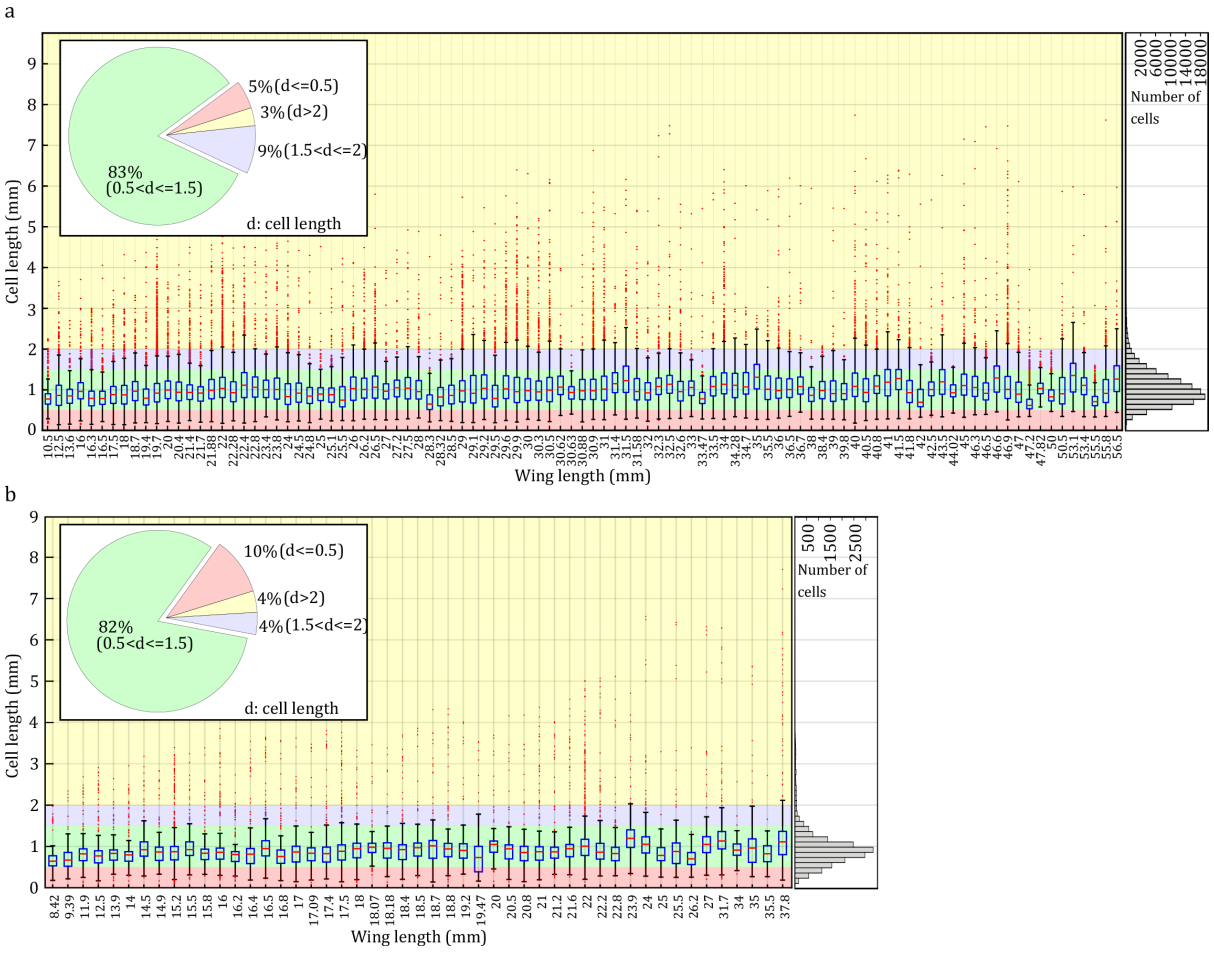



Figure 7.2: Distribution of cell length in dragonflies (a) and damselflies (b) categorized by wing length. Box plots showing the median, interquartile range, and outliers of cell lengths for each wing size category. Pie chart depicting the proportion of cells falling within different length ranges. Histogram displaying the frequency distribution of cell length ranging from 0.1 to 9mm.

wing. However, it seems the results for comparing the minimum cell size and the wing size in damselflies are not statistically significant due to having a P-value of 0.06. Table 7.1 shows a correlation of 0.76 and 0.84 between the wing area and the maximum cell length in the wing of dragonflies and damselflies, respectively. This shows that the length of the largest cell increases proportionally with wing size (Figure 6.3a.2, b.2). Notably, our findings reveal that larger cells tend to be located near the margins of both dragonfly and damselfly wings, as depicted in panels a.3 and b.3 of Figure 7.3. Moreover, we observed that the biggest cells are usually situated in the basal part of the wings. Besides, Figure 7.3a.3 shows in dragonflies almost more than 99% of cells located at a distance more than 3 mm from margins having a length of less than 2.5 mm. This value in damselflies (Figure 7.3b.3) is 98% for cells with a length less than 1.5 mm and with a distance more than 1mm from the wing margin. Further examination of panels a.4 and b.4 in Figure 7.3 reveals that smaller cells tend to be more circular than larger cells, which exhibit more significant variation in small cells' shape. A correlation of -0.64 between the cell length and the circularity in damselflies and a correlation of -0.58 in dragonflies shows a reverse trend between the cell length and circularity. However, the relationship has moderate strength.

| | | Dragonfly | | Damselfly | |
|---|---|---|---|---|---|
| Variable Pair | | Correlation | P-Value | Correlation | P-Value |
| Wing Area | max(Cell Length) | 0.76 | << 0.001 | 0.84 | << 0.001 |
| Wing Area | median(Cell Length) | 0.46 | << 0.001 | 0.37 | << 0.001 |
| Wing Area | min(Cell Length) | 0.56 | << 0.001 | 0.18 | 0.06 |
| Wing Area | Number of Cells | 0.77 | << 0.001 | 0.89 | << 0.001 |
| Cell Length | Cell Circularity | -0.58 | << 0.001 | -0.64 | << 0.001 |

Table 7.1: Correlation analysis between wing characteristics and cell properties in dragonflies and damselflies.

The cellular structure of insect wings is highly complex and heterogeneous. Insect wings consist of a variety of cell types, each with its unique structure and function. In addition, the size and shape of these cells can vary widely, depending on their location within the wing and their role in maintaining structural integrity [43, 118]. One of the most interesting findings of this study is that most of the cells have a length between 0.5 and 1.5 mm, corresponding to the critical crack length of locust wings' membrane [8, 196]. It is noteworthy that microscopic observations reveal irreparable small cracks in various parts of the wing [181, 197]. The cellular structure of the wing appears to be a strategy aimed at delaying wing failure. Furthermore, our discoveries indicate that the wing cell length could be optimized to reduce the possibility of cracks and other types of mechanical damage. Nevertheless, it has been previously established that cross veins have an adverse effect on the wing's natural frequency [13].

In our previous investigation, we gathered 119 wings from over 30 individual *Sympetrum vulgatum* dragonflies to identify which sections of the wing experience more damage [73]. All samples were collected during the latter half of their lifespan while in flight. The results indicate that the probability of damage occurring in the trailing edge and wing tip is higher compared to other regions, likely due to their increased exposure to obstacles during flight. Our findings reveal that the cells in these specific areas exhibit smaller lengths. As a result, we can infer that the cellular structure of the wing could potentially endure wing cracks that occur at the trailing edge or tip, providing a level of resilience in these vulnerable regions. Furthermore, our findings suggest that the basal

regions of the wing contain larger cells, which are known to experience stronger aerodynamic forces [198]. Consequently, one would expect these areas to exhibit more damage. Surprisingly, our observations of damaged wings [73] contradict this expectation. However, it is important to note that we cannot definitively conclude that damage never

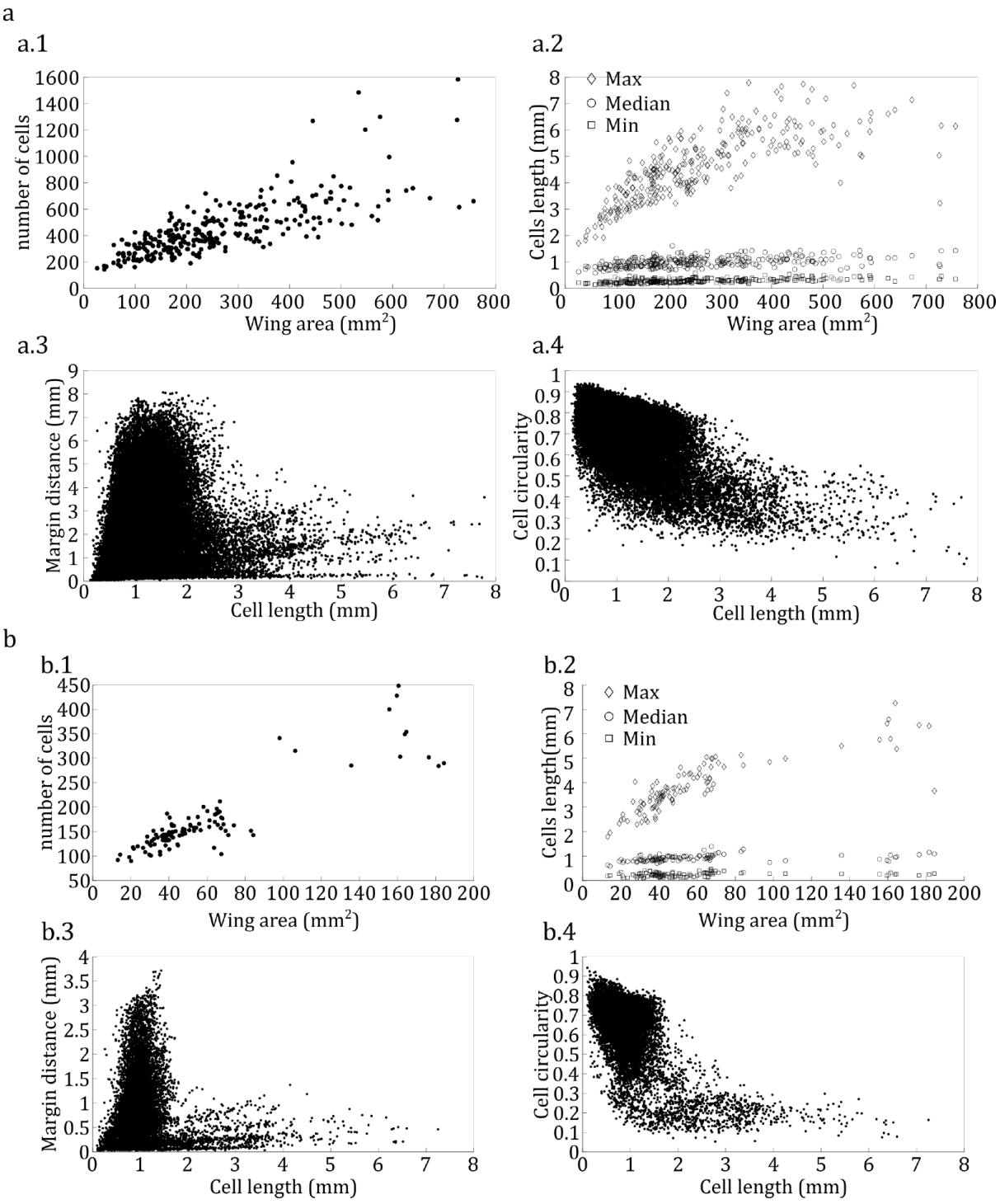



Figure 7.3: Statistical data on the size and geometry of insect wings for dragonflies (a) and damselflies (b). Each panel contains four scatter plots depicting (1) the relationship between the number of cells and the area of the wing, (2) changes in the maximum, minimum, and median of cell length with wing area, (3) changes in the cell length based on the distance from the wing margins, and (4) changes in the cell circularity with cell length.

occurs in the basal areas [74]. It is worth considering that the presence of thicker veins and corrugations may contribute to enhancing the resilience of these basal parts.

Moreover, the circularity of cells is crucial in maintaining the cell size in all directions, resulting in the maximum area while using less material [118]. In areas of the wing apart from the basal part and leading edge, the thickness of the veins is reduced, a crucial factor in balancing flexibility, stiffness, and lift generation. Although thinner veins may risk increased damage, having smaller-sized cells enhances the resilience of these wing sections. In such instances, the membrane area is compromised if the cells lack a more circular shape. More cells are needed to compensate and maintain sufficient area, resulting in increased weight. As a result, we assume this is why smaller cells have more circular shapes. It is important to mention that we must consider other wing components while striding the size of cells. The cuticle thickness is non-uniformly distributed among the wing [76]. Also, the wing itself is not a planar structure, but it is entirely corrugated, and the corrugations strongly influence the structural stiffness of the wing [139, 199]. The relationship between the spatial shape of the wing, the cell size, and the cuticle thickness remains elusive and yet to be understood.

The wing functionality is also affected by its microstructure, which is another crucial factor. Prior research has shown that the wing contains two distinct types of micro joints: fused and flexible [198, 200]. Fused micro joints are characterized by veins that intersect and come into significant contact with each other, while flexible micro joints have minimal direct contact between veins and are typically connected by a section of resilin-rich cuticle [33, 201]. It is also known that the presence of flexible micro joints containing resilin in insect wings can decrease the likelihood of material failure by avoiding stress concentrations at intervals between veins [10]. In addition, flexible microjoints are seldom observed in regions with smaller cells [118], suggesting that cell size may be a compensatory mechanism for the absence of these joints.

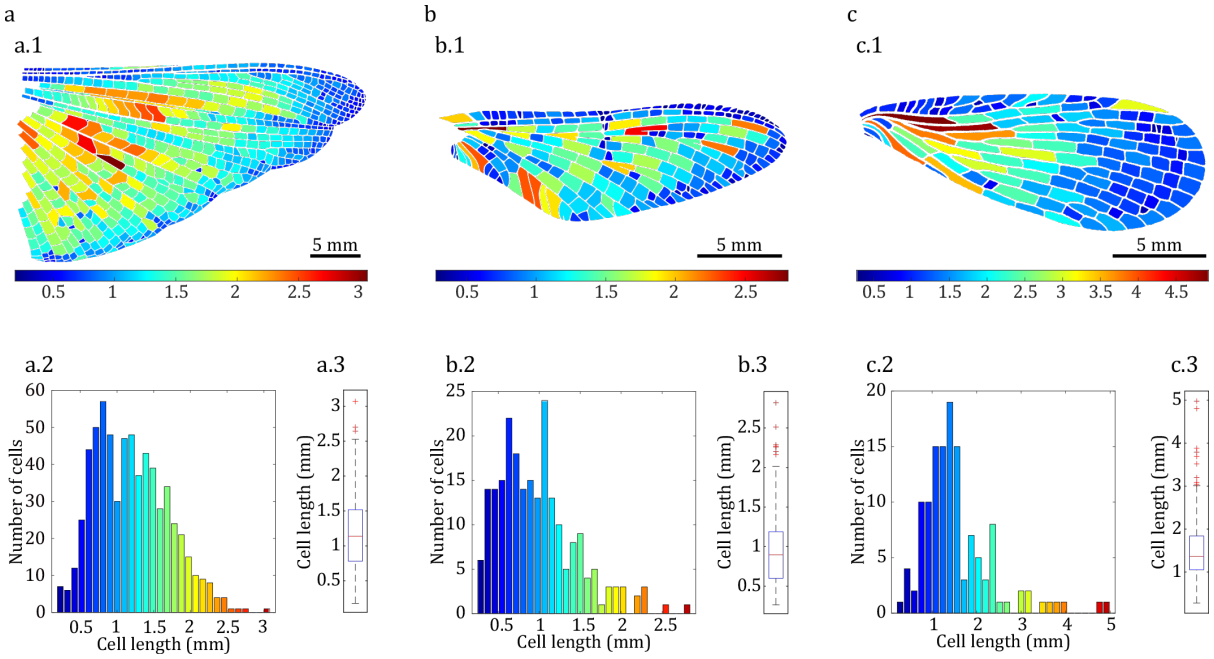



Figure 7.4: Cell length distribution in three different insects from three different orders. Subpanels (1), (2), and (3) in each panel show the contour plot, histogram, and box plot of cell length distribution, respectively. Panel (a) represents *Caelifera* (dessert locust), panel (b) represents *Ephemeroptera* (mayfly), and panel (c) represents *Mecoptera* (scorpionfly).

The current study suggests that the size of cells on the Odonata wing might be a way to prevent cracks from propagating, but further research can reveal if this is valid for other types of insects. However, significantly larger cellular structures are common in many insects. To explore this further, the wings of a desert locust (Figure 7.4a), mayfly (Figure 7.4b), and scorpionfly (Figure 7.4c) were examined. The results indicate that the trend observed in the Odonata wing is also present in these insects, with many cells having a length between 0.5-1.5mm. This finding provides additional evidence that the cellular structure is a strategy used by many insects, not only Odonata, to maintain the functionality of their wings and extend their lifespan. Ultimately, it is essential to note that insect wings are crafted not just to withstand damage but also to generate lift [202]. Various elements interact simultaneously to maintain their functionality, such as micro joints, composite materials, vein thickness, corrugation, and various vein junctions [14, 26, 33, 75, 76]. This research specifically focuses on the wing's morphometry, particularly the cell sizes, to explore their role in enhancing wing durability. We assert that further in-depth studies are necessary to uncover the multi-functionality of insect wings. The data presented in this study is valuable and can contribute significantly to the exploration of insect wings through the application of multi-objective optimization, machine learning, and deep learning methods.

## 7.5 Conclusion

In this study, we found that regions with smaller cells tend to lack flexible joints and instead rely on cell size as a compensatory mechanism to maintain wing integrity. In contrast, regions with larger cells often feature flexible joints that prevent stress concentrations and reduce the risk of material failure. Interestingly, we also observed that the length of cells located in the farthest regions from the wing margins is in the critical crack length range. This suggests that there may be a functional trade-off between cell size and wing size, with smaller cells providing more excellent resistance to crack propagation at the expense of higher material costs and higher local weight. To sum up, examining the cellular makeup of insect wings offers valuable insights into how nature has evolved to optimize strength, flexibility, and damage resistance. By creating biomimetic designs based on studying these structures, new materials and systems that possess unique properties and can be utilized in various fields can be produced. Our research has potential implications for the design of synthetic wings, crack-resistant meta-materials, and thin films [203]. By comprehending how cells are distributed according to their specific characteristics in insect wings, we can devise better methods to imitate their structural features and improve the longevity of engineering systems for a broad range of applications.

## Sepplementary Files

All data for this study are documented in the Zenodo repository via the following link [195]. A 'Supplementary Information' file is also available, which provides the structure of documented files and access links to WingSegment and WingGram.
https://doi.org/10.5281/zenodo.10557201

# Chapter 8

# General Discussion

## 8.1 Advancement in the Study of Insect Wings

This thesis delved into the advancement of novel methodologies aimed at enhancing the speed, efficiency, and precision of insect wing investigation. Four distinct software packages—WingMesh [59], WingGram [60], WingSegment [77], and WingAnalogy—were developed and are individually described, and discussed in Chapters 3, 4, 5, and 6. This chapter will comprehensively discuss all the developed approaches. Table 8.1, represents a side-by-side comparison between the developed software packages.

| **Tools and Options** | **WingAnalogy** | **WingSegment** | **WingGram** | **WingMesh** |
|---|---|---|---|---|
| Abaqus INP Model | No | No | No | Yes |
| Abaqus JNL Model | No | No | Yes | No |
| FreeCAD Model | No | Yes | No | No |
| Cell Area | Yes | Yes | Yes | No |
| Cell Length | Yes | Yes | Yes | No |
| Cell Width | Yes | No | Yes | No |
| Cell Circualrity | Yes | Yes | No | No |
| Cell Centorids | Yes | Yes | Yes | No |
| Cell Distance from Margin | No | Yes | No | No |
| Voronoi Pattern Generation | No | Yes | No | No |
| Junction Detection | Yes[a] | Yes[a] | Yes | No |
| Venation Pattern Detection | No | Yes | No | No |
| Pterostigma Detection | No | Yes | No | No |
| Complex-Compatible | Yes | Yes | Yes | No |
| 3D Printable Model | No | Yes | Yes | No |
| Corrugation Assignment | No | No | Yes | Yes |
| Automated Superimposing | Yes | No | No | No |
| Asymmetry Analysis | Yes | No | No | No |
| Mass Analysis | Yes | No | No | No |
| Report Generator | Yes | No | No | No |

[a]Improved and Modified

Table 8.1: Comparison between WingAnalogy, WingSegment, WingGram, and WingMesh

### 8.1.1 Segmenting Insect Wing Image

In this thesis, WingSegment [77] (Chapter 5) presents new automated methodologies for segmenting insect wings. The results and validations not only demonstrate high levels

of efficiency, precision, and speed but also highlight the applicability of the developed methods across diverse wings with varying geometric complexities (See Section 5.3.1). Tools in WingSegment specifically extract the boundary of cells, the outline of the wing, the place of junctions, the vein path, and if available the place of Pterostigma with less effort.

While ImageJ is a well-known software with tools for segmenting images of so-called cellular structures, such as insect wings, it requires users to adjust various settings manually for the best results [61]. It's important to note that ImageJ is a powerful imaging tool with broad capabilities. However, when it comes to insect wings, WingSegment outperforms it in terms of automation and speed. DrawWing [56] is another tool designed for extracting junctions and creating drawings of wing veins and outlines. However, it is specifically tailored for the study of honeybees belonging to the genus *Apis* and may not be as versatile for other insect species. NET (Network Extraction Tool) [65] and NEFI (Network Extraction from Images) [66] are alternative tools used for extracting junctions and networks from 2D cellular images, such as insect wings. Although they are known for their speed and efficiency, it's essential to note that they focus on extracting connections between junctions rather than the complete network path. In contrast, WingSegment's embedded tools not only accurately extract junctions but also capture the vein paths between these junctions. FijiWings is another tool designed for quantifying trichomes and junctions in *Drosophila*, and is not applicable for other wings, while still, WingSegment proved to be applicable for diverse wings.

### 8.1.2 Morphology Analysis of Insect Wings

The automated morphological analysis of insect wings represents another advancement in this thesis. The developed tools provide statistical information through quantities, boxplots, and distribution contours for wing cell area, length, width, circularity, vein length, placement and number of cells and junctions, and the distance of cells from wing margins.

WingGram (Chapter 4), WingSegment (Chapter 5), and WingAnalogy (Chapter 6) all provide morphology analysis tools. Notably, WingSegment, developed after WingGram, introduces additional features such as vein path detection and wing cell circularity measurement, enhancing the capabilities inherited from WingGram. WingAnalogy, on the other hand, is particularly recommended for studying asymmetry in insect wings. Besides, compared to other tools like DrawWing, and FijiWing, WingSegment is applicable for a wide variety of insect wings from simple to complex ones [56, 57]. WingSegment also shows a high robustness which was discussed in Section 5.4.5.

Using this advancement, in Chapter 7, the morphology of 389 wings from 197 species of Odonata was investigated. The analysis revealed a significant variation in cell sizes, ranging from less than 0.5 mm to about 8 mm. Notably, more than 80% of the cells fell within the size range of 0.5 to 1.5 mm. Cells located farther from the wing margins predominantly fell within the 0.5 to 1.5 mm range, while smaller cells were primarily situated on the margins. In a prior study on locust wings, a critical crack length of 1 mm was identified for the membrane, and most cells were found to have a length of about 1 mm [8]. The study concluded that veins are strategically positioned to reinforce the wing by forming a cellular structure with sizes within the critical crack length range, thereby enhancing the wing's resilience to sudden failure. A similar procedure is proposed in the current study on Odonata wings, suggesting that the observed cellular structure serves a comparable purpose in reinforcing the wings.

Automated morphometry analysis is crucial for intricate structures such as insect

wings, particularly in the case of complex wings found in insects like Odonata, Locusts, and Stick Insects [8, 55, 204]. Due to the lack of suitable methods, the literature indicates more studies have been conducted on less complex wings, while complex wings offer more interesting features to be investigated. Its importance lies in minimizing human error, ensuring the reproducibility of data, and achieving greater precision by eliminating human manipulation in measurements and reducing the impact of vision errors.

### 8.1.3 Finite Element Modeling of Insect Wings

This thesis addresses challenges due to the 3D modeling of insect wings by introducing three software packages with distinct tools for finite element (FE) modeling of insect wings in Chapters 3, 4, and 5 [59, 60]. The automated segmentation method, developed through fast marching and region growing [71, 72, 79, 80], facilitates the process of extracting boundaries to create wing models even without having skills in 3D modeling.

In WingMesh, distmesh2d is employed to generate meshes based on the extracted geometry from the image [99]. This approach not only advanced the precise and automated 3D modeling of insect wings but also improved distmesh2d by enabling the definition of the arbitrary domain from an image. WingMesh has three major limitations.

I. The processing speed significantly decreases, especially as the number of cells increases and the domain becomes more complex.

II. The final model consists of generated meshes, making it challenging to edit the model's geometry easily.

III. The user lacks control over both the size and type of the generated mesh.

WingGram was subsequently developed to address these challenges by constructing wing models through Python scripting in Abaqus using the journal (JNL) file format.

#### Corrugation Assignment

The tool embedded for the assignment of corrugations within WingMesh and WingGram represents another advancement in the finite element modeling of insect wings. However, this needs the preparation of a secondary image (see Sections 3.2.4, and 4.2.3) and is not as accurate as a 3D scanner but still, it offers advantages. 3D scanning of insect wings requires specialized 3D scanners, which are primarily expensive and not easily accessible [205]. The reflectivity and transparency of wing models may pose challenges to the scanning process, necessitating specialized equipment and techniques. Additionally, the substantial volume of data generated by 3D scanners can be computationally intensive [139, 161, 206].

Despite the advantages of speed and accessibility, assigning corrugations using a secondary image offers distinct benefits. As depicted in Figure S3, the secondary image can be edited to define various corrugations. In the mentioned figure, three different secondary images led to models featuring only concavity, another with both corrugation and concavity and the last one solely with corrugation and without concavity. This provides new avenues for exploring the impact of corrugations from both aerodynamic and structural behavior perspectives [12, 13, 120]. Moreover, this approach may contribute to the optimized design of corrugated artificial wings for flapping wing robots [207, 208].

### 8.1.4 Asymmetry Analysis of Insect Wings

Numerous studies have been conducted on the measurement of asymmetry in insect wings using landmarks [45, 53, 167, 209, 210]. However, in this method, abnormal wings are excluded due to the requirement for a consistent number of landmarks in all study cases. Moreover, in complex wings, such as those from the order Odonata, many cells and veins are excluded, and a limited number of landmarks are considered for measuring asymmetry [51]. WingAnalogy employs computer vision to segment insect wing images, and then it compares wing pairs by analyzing various parameters, including area, length, width, circularity, and the position of corresponding cells. Additionally, the average distance between corresponding junctions, the number of cells and junctions, the area, perimeter, and length of wings, and the average distance between wing outlines are considered. WingAnalogy addresses the challenge of wing superimposition by automating the process through Particle Swarm Optimization (PSO). This optimization technique repositions one wing to minimize the distance between the outlines of the wings. WingAnalogy presents advantages in terms of higher speed, scalability, precision, and reliability compared to conventional methods. Notably, human intervention is minimized in the process of computing asymmetry within this tool. Furthermore, WingAnalogy allows each wing to be divided into a maximum of five regions, facilitating localized investigation of wing asymmetry. This tool is used to analyse White-legged damselfly *Platycnemis pennipes*, and Stick Insect *Malacomorpha cyllarus*.

#### Asymmetry Analysis of White-legged damselfly *Platycnemis pennipes*

In this study a total of 28 males and 30 females of *Platycnemis pennipes* were collected, and their wings were carefully dissected and scanned using an Epson Perfection V850 Pro. Each individual contributed two forewings and two hindwings to the study.

The primary objective of this research was to explore the frequency of asymmetry within four distinct groups for both males and females, considering the entire wing. These groups included forewings, hindwings, right wings, and left wings.

Furthermore, the specialized tool developed in WingAnalogy, designed for splitting wings into a maximum of five sections, was employed. This tool facilitated the division of wing pairs into two, three, and five different cell sets vertically. Additionally, it divided the wings into three sets horizontally.

Collecting such comprehensive data for investigating insect asymmetry without the aid of WingAnalogy would have been a time-consuming and error-prone task. This underscores the value of computer vision and similar algorithms in automating the analysis of wing images.

This study demonstrates that the hindwings of male *Platycnemis pennipes* are generally more symmetric in shape compared to those of females. The closest symmetry is found at the wing's base, where cell preservation ensures structural stability. Similarly, the coastal margin is more symmetric than the posterior margin[1].

#### Asymmetry and Morphometry Analysis of Stick Insect *Malacomorpha cyllarus*

The stick insect *Malacomorpha cyllarus* possesses relatively large wings compared to other insects, featuring an intricate shape. Figure 8.1 shows one edited image of this insect.

[1]This study was conducted as part of Lisa Claußen's Master thesis at Kiel University, under the supervision of Prof. Gorb and myself [211].

In this image, approximately 570 cells, 1040 junctions, and 1616 veins are detected. Studying the morphology of such structures is prone to human error and time-consuming with conventional methods. Therefore, WingAnalogy and WingSegment were chosen to extract the geometric features of these wings and investigate their asymmetry.

For this study, 20 wings were scanned using an Epson Perfection V850 Pro scanner. The research focused on examining the distribution of cell area and vein length and investigating the asymmetry of the wings. Additionally, the wings were divided into three cell sets to explore asymmetry at a local level[2].

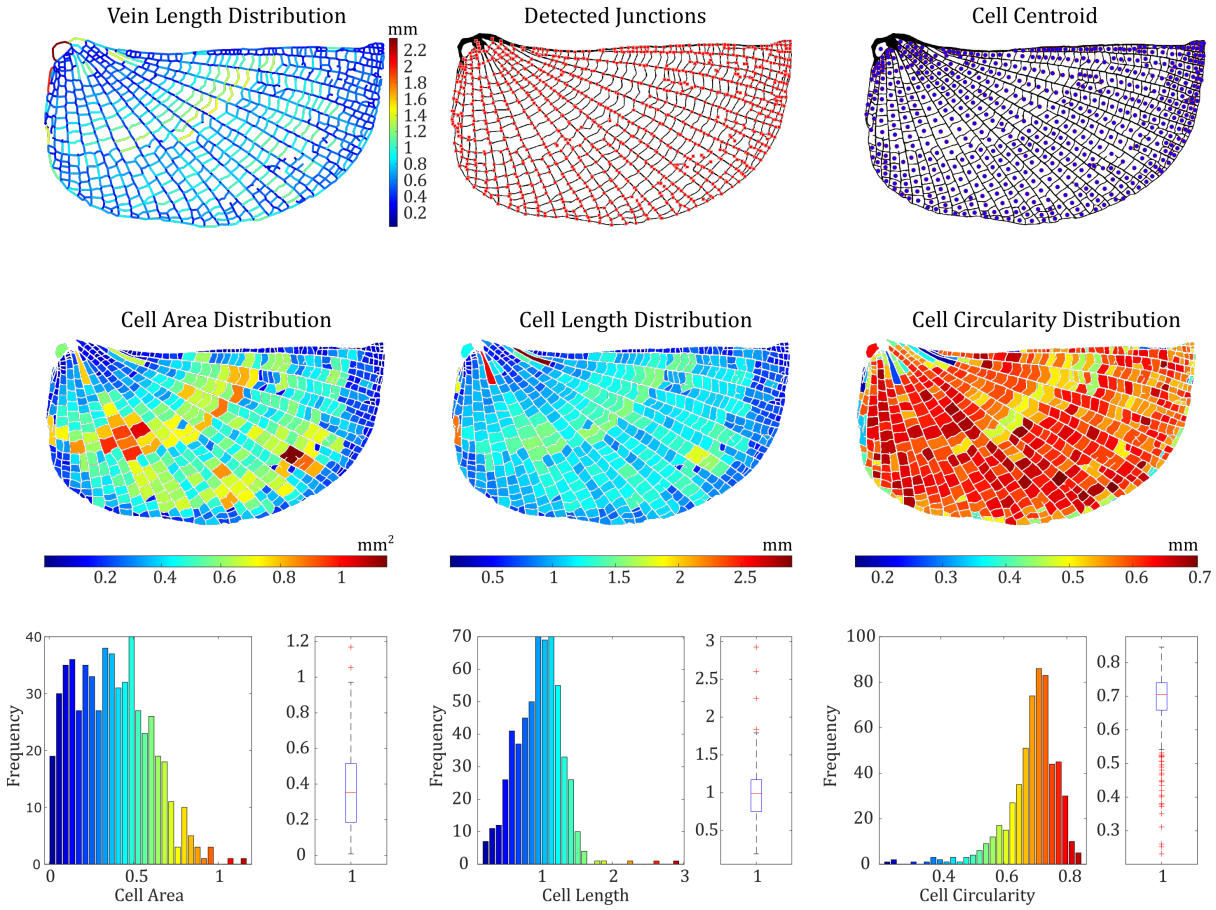



Figure 8.1: Extracted cells, veins and junctions from the stick insect *Malacomorpha cyllarus*

## 8.2 Constraints and Limitations

### 8.2.1 Image Preparation: Issues and Limitations

Although developed methods demonstrate efficiency in analyzing and 3D modeling insect wings, there are limitations. Preparing images for these tools requires human intervention, leading to the first limitation: not all images are compatible with the developed tools. Specifically, wing images with noise, unclear veins, and very thin veins yield inefficient results. Furthermore, these methods do not work efficiently with wings that have dark spots unless the image is edited to meet the criteria described in Chapter 2.

[2]This study was conducted as part of Luise Kittner's Master thesis at Kiel University, under the supervision of Prof. Gorb and Dr. Büscher in collaboration with myself [212]

Consequently, these tools are not effective for analyzing the wings of species such as the Black Saddlebags Dragonfly (*Tramea lacerata*), Common Whitetail Dragonfly (*Plathemis lydia*), Ebony Jewelwing (*Calopteryx maculata*), Death's-head Hawkmoth (*Acherontia atropos*), Peppered Moth (*Biston betularia*), Mourning Cloak Butterfly (*Nymphalis antiopa*), Blue Morpho (*Morpho peleides*), Carpenter Bee (*Xylocopa*), Blue Mud Dauber (*Chalybion californicum*), and Black Horse Fly (*Tabanus atratus*).

### 8.2.2 3D Modeling Limitations

However, the developed tools for 3D modeling of wings in this study are novel and do not have similar rivals, but still, they have several limitations that can be improved in the future. Developed methods generate the model from a 2D image, by extracting the geometry of cells which results in shell models. This means in generated models the gradient of thickness, and the geometry of cross-sections are not considered. Moreover, other components of the wing, like micro joints, spikes, nodus, or locking mechanisms are not involved in developed models.

The assignment of corrugations, as detailed in Sections 3.2.4 and 4.2.3, requires the preparation of a secondary image, which presents certain challenges. This process necessitates spatial information of the wing that identifies the locations of hills and valleys. Consequently, this requirement limits the tool's ability to generate corrugations for extinct species based solely on fossil images. However, it is possible to infer this information to some extent from similar species within the same order. In the ideal scenario, possessing a 3D scan of the wing facilitates the preparation of the secondary image. If the question arises as to why one should generate a 3D model using the developed tools in this thesis when a 3D scan is available, the answer lies in the limitations of 3D scanning technology. A 3D scanner cannot differentiate between cells and veins, whereas the WingMesh and WingGram tools, used for generating a corrugated model, offer significant advantages. Not only can cells and veins be modeled separately, but there is also the option to define each cell individually, a capability not yet possible with 3D scanners. This distinction highlights the specialized functionality of WingMesh and WingGram in capturing detailed anatomical features of wings that are crucial for precise scientific analysis and modeling [59, 60].

### 8.2.3 Morphology Analysis Limitations

In this thesis, we have not developed an algorithm for counting trichomes on the wings of insects such as *Drosophila* [57]. The current methodologies do not allow for the measurement of vein thickness or the angle between veins. Additionally, the developed methods are unable to distinguish between different parts of the wing, such as the base, leading and trailing edges, and tip, treating these instead as part of the general vein structure. The location of nodus in Odonata is also not detectable with the current tools [103]. Furthermore, the developed methods cannot recognize longitudinal veins. These limitations present opportunities for future research and development to enhance the accuracy and functionality of the tools for detailed anatomical analysis of insect wings.

## 8.3 Beyond Entomology

All methods developed in this thesis are dedicated to insect wings but can cover a broader range of subjects. These methods operate based on images of insect wings, with the

algorithms extracting features due to the cellularity of the wings, despite having no inherent knowledge of the wing itself. Consequently, these methods apply to all other 2D cellular structures, whether they originate from natural or artificial tissues. Such structures could include cross-sections of trees and bones, images of clay and soil cracks, spider webs, the cellular structure of leaves and foams, as well as microscopic images of cells and bacteria. Figures 5.4, and and S2 depict some examples.

## 8.4 Better Than Artificial Intelligence?

Artificial Intelligence (AI) is the branch of computer science focused on creating systems capable of performing tasks that normally require human intelligence. Some studies have been conducted on insect wings using AI, deep learning (DL), and machine learning (ML) for classification, identification, and monitoring of insects [62, 213–217]. AI needs to be trained to achieve specific tasks with high precision. However, the variety of insect wings and the lack of a big database of insect wing images pose challenges to this issue [218]. That is the reason why many studies employing AI for the analysis of insect wings mainly focus on training models from less complex wings, such as those from *Drosophila*. However this is a very big claim that AI is not efficient, but some reasons make classical computer vision in this specific case (insect wings) more effective.

- **Data Quality and Availability**. High-quality databases are crucial for training accurate ML, and DL models. In the context of insect wing studies, collecting this amount of data is very time-consuming and needs expert knowledge. However, collecting large data from extinct species is not possible which makes it difficult in the study of fossils [218–222].
- **Variety in Shape and Size of Wings**. AI and ML models may struggle to generalize across different species of insects or different morphological variations within a species. This is made worse by biases in the data used to train them, which may not cover all the different shapes, sizes, patterns, and conditions of wings found in nature [42, 43].
- **Ethical Concerns**. Collecting data from insect wings raises ethical concerns because it requires sacrificing a large number of insects to gather sufficient data. In some cases, this practice may endanger certain species.

The developed methods in this study for analyzing insect wings using classical computer vision methods rely on image quality and do not require a database for extracting morphological features from insect wings. However, they do require human intervention to prepare the wing image. Nevertheless, they can still be efficiently utilized in studying wings from various species, even those with very high complexity levels. Additionally, this method, owing to its inherent generalizability, can be applied to other cellular structures such as leaves and foams, or for the study of clay crack development.

# Chapter 9

# Ongoing Projects and Future Directions

As this thesis has explored the development of new methodologies for the study of insect wings, the natural progression of this research can contribute to the design and development of technological structures inspired by biological systems. One significant advancement of this research is the development of automated modeling methods, which facilitate the preparation of 3D printable models of insect wings. Several studies have used 3D printing or laser cutting to manufacture artificial wings [21, 223–226]. In these studies, the mechanical behavior of artificial wings, followed by their aerodynamic behavior, has been studied to design lightweight wings with high efficiency in terms of structural stiffness, flexibility, lift generation, and drag reduction. Having WingSegment (Chapter 5) and WingGram (Chapter 4) facilitates the 3D model generation of wings across various species, including those with complex geometries. Furthermore, drawing on the findings from Chapter 7, we've gained valuable insights about the cellularity of insect wings, providing further guidance on designing artificial structures inspired by insect wings. Currently, several projects are underway based on findings from this thesis, as outlined in this chapter.

## 9.1 Design of Durable Bio-Inspired Thin Cellular Structure

In this research, seven cellular thin models have been designed, all with approximately the same number of cells (Figure 9.1f-l). These cellular structures have been generated using Voronoi patterns derived from customized centroids. The centroids have been generated using a custom MATLAB code. The code generated the images depicted in Figure 9.1a-d, and then WingSegment was used to make the models from the generated images. In Figure 9.1a, the density of cells increases from bottom to top, while in Figure 9.1b, the density increases from both the bottom and the top towards the middle. In Figure 9.1c, and d, the gradient of size is similar, but the positions of centroids are adjusted to generate squares in Figure 9.1c and hexagons in Figure 9.1d. Figure 9.1f,g and h are Samples I, II, and III generated from the image in Figure 9.1a. Figures 9.1i, and j shows samples IV and V generated from the image in 9.1b. Figure 9.1k, and l respectively illustrate Samples VI, and VI generated from images in 9.1c, and d. These models are 3D printed once with a thin membrane of paper, and once without a membrane.

Two questions are behind this research. i) How does the heterogeneous size distri-

bution of cells affect the durability of the structure in both tension and torsion loading? ii) Does a thin membrane contribute to the durability? To clarify the second question, it remains unknown whether the membrane serves as a sealed area contributing to lift generation or if it primarily influences the structural stiffness of the wing [8, 13, 227, 228].

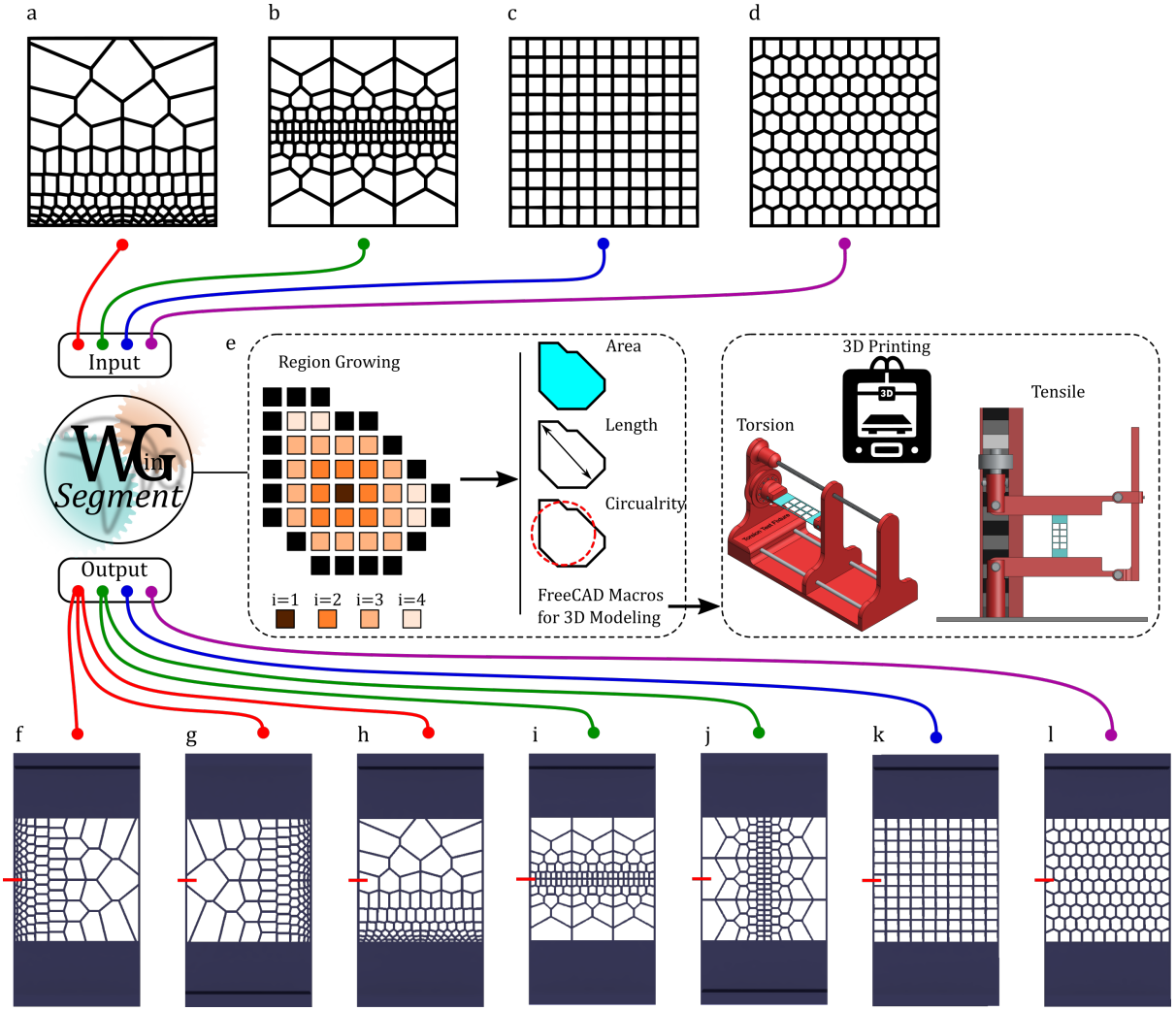



Figure 9.1: Comprehensive visualization of thin model generation using WingSegment.(a-d) Images produced by a custom MATLAB code, illustrating the initial models inspired by insect wings. (e) shows the methodology adopted for both experimental procedures and modeling techniques. (f-h) present samples 1, 2, and 3, respectively, derived from the model in panel (a). (i) and (j) showcase samples 4 and 5, based on the model in panel (b). (k), and (l) respectively show sample 6, and 7 generated from panels (c), and (d).

For manufacturing the models, a fused deposition modeling (FDM) 3D printer (Prusa i3 MK3S, Prusa Research, Praha, Czech Republic) was utilized. Polylactic acid (PLA) served as the printing material, with a nozzle size of 0.25mm and a fixed temperature of 215°C for the nozzle and 60°C for the bed. To apply the membrane to the model, paper was adhered to the bed, allowing the printer to directly print the models onto the paper, ensuring perfect adhesion without the need for glue. For the torsion and tensile tests as depicted in Figure 9.1e, two fixtures were designed to be installed on a ZwickiLine uniaxial testing machine (Zwick Roell, Ulm, Germany), equipped with a 500 N load cell (Xforce P load cell, Zwick Roell). The results of this study indicate that the presence of a membrane significantly increases the structural stiffness of printed samples, while the weight increase is minimal. Furthermore, the results demonstrate a significant effect of

the cell size gradient on the mechanical behavior of the structure. We anticipate that this research will also provide insights into the effects of cell size and membrane on the mechanical behavior of insect wings.

## 9.2 Study the Effect of Corrugation on the Mechanical Behavior of Insect Wings

WingGram's tool simplifies the assignment of corrugations, aiding in a systematic examination of their function in insect wings. As illustrated in Figure S4, three wing models are generated: one with concavity, another with corrugations, and a third that combines both features. The primary objective of this study is to evaluate the mechanical behavior of these models and compare them to a reference model that lacks any corrugations or concavity. Currently, this investigation is undergoing finite element analysis.

## 9.3 Manufacturing of 3D Corrugated Wing through 3D Printing and Thermal Post-Processing

In this doctoral research, a method for finite element (FE) 3D modeling of insect wings has been developed. This method incorporates the assignment of corrugations to the 3D model using a secondary image. This approach not only yields 3D models that are more realistic due to the incorporation of corrugations but also opens new avenues in the study of the spatial geometry of insect wings. However, while the models generated by this method can be exported as an STL file for 3D printing, the process of 3D printing such complex, fine structures is not straightforward for all 3D printers.

A new approach is currently under development for manufacturing corrugated 3D-printed wings through thermal post-processing. In this method, a wing is first 3D printed using PLA filament without corrugations. For generating a 3D printable model, WingSegment can be utilized. Male and female molds, as depicted in Figure 9.2f, are then used to shape the wing under controlled conditions. These molds represent the desired corrugations. Before being placed between the molds, the printed wing needs to be warmed to 60°C. This temperature softens the PLA, making it malleable. Subsequently, the wing is pressed between the molds under warm water and then allowed to cool down. The molds must be made from a material with a higher melting temperature, such as ABS, to prevent them from softening during the process. The development of the molds necessitates a programming approach that automatically generates the corrugations using the same method as applied to the finite element (FE) wing described in Section 4.2.3.

## 9.4 Future Directions

- **Advancements in Automated Modeling.** The methodologies presented in this thesis for the automated modeling of insect wings could be enhanced by exploring additional possibilities. This may involve incorporating the vein pattern with customized cross-sections and integrating gradients of thickness and material properties into the developed models. Furthermore, improvements could be made in the assignment of corrugations.

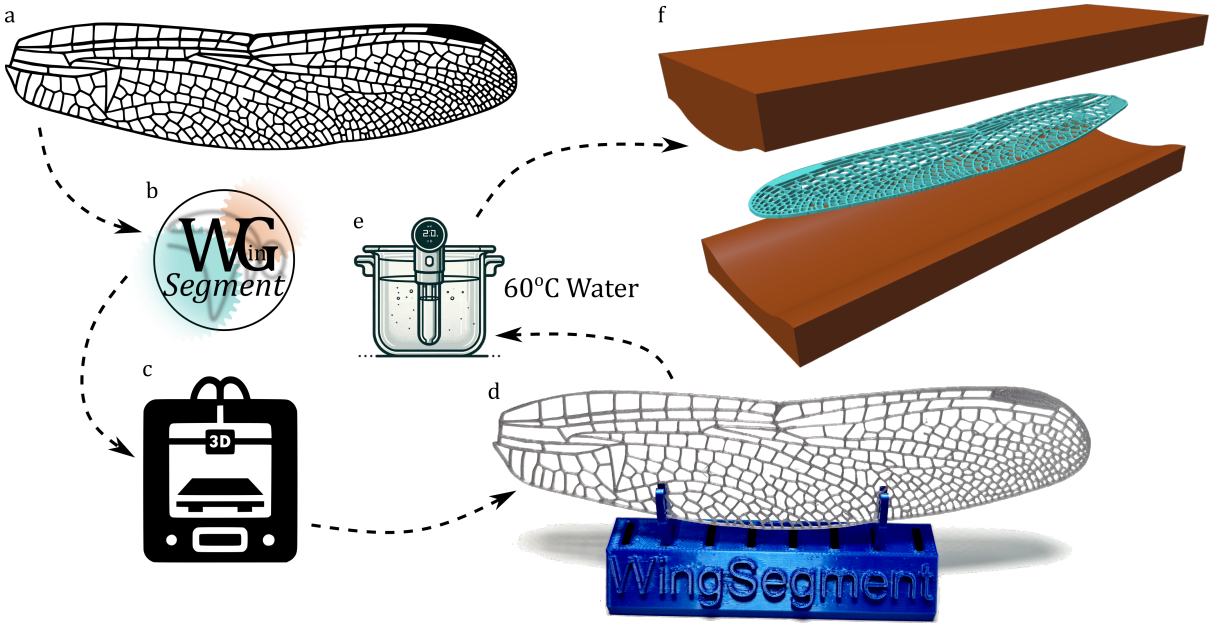



Figure 9.2: Procedure for Assigning Corrugations to a 3D-Printed Insect Wing through Thermal Post-Processing

- **Refinement of Asymmetry Analysis.** There is potential for enhancing asymmetry analysis by incorporating additional statistical measures such as analysis of variance, T-test, and correlation analysis. Moreover, improvements can be made in the computation of both fluctuating and directional asymmetry.
- **Investigate the Correlation Between Wing Cell Dimensions and Overall Wing Sizes Across Various Insect Species.** This study explored the relationship between wing cell size and crack propagation. A similar investigation could be extended to other insect wings to determine if comparable patterns exist. Additionally, experimental tests could be conducted to gather further insights into the critical crack length in Odonata wings and potentially in the wings of other species.
- **Design and Development of Artificial Wings.** The developed automated modeling methods can be utilized to extract key features of insect wings, inspiring the design of bioinspired wings. Additionally, using 3D modeling and printing, we can translate these features into physical prototypes, optimizing for aerodynamics and structural integrity. This process enables the rapid development of innovative wing designs for applications in robotics, aerospace, and beyond.

# Chapter 10

# Conclusions

I. This doctoral research explored the employment of image processing (IP) and computer vision (CV) techniques to significantly advance the study of insect wings. State-of-the-art IP and CV algorithms were designed, which helped considerably with the automated segmentation and morphological analysis of insect wings. These methods have streamlined the detection of complex wing patterns and provided new insights into the biological and functional significance of those patterns.

II. This research opened up the way for automated 3D modeling for finite element analysis and 3D printing of insect wings from digital images. This approach simplifies and speeds up the modeling process, while reducing human errors, and offers editability, reproducibility, and scalability.

III. This thesis addressed the challenge of corrugation assignment in the Finite Element modeling of insect wings. This contribution marks a significant step towards understanding the function of corrugations in the mechanical resilience and aerodynamic efficiency of insect wings.

IV. Building upon these advancements, the research also addressed the challenge of asymmetry analysis in insect wings, providing a more reliable, scalable, fast, and precise method of evaluation. This approach simplifies the asymmetry analysis of more complex wings, such as those from Odonata, Locusts, or stick insects, by offering automated superimposing and asymmetry measurements.

V. Having automated tools simplified the investigation of the relationship between wing and cell size in diverse families of Odonata using 389 wings. These have been the study of patterns that could be critical to large wing structural integrity and, in this sense, would provide insight into adaptations for crack resistance.

# Bibliography


[1] H Rajabi and SN Gorb. How do dragonfly wings work? A brief guide to functional roles of wing structural components. *International Journal of Odonatology*, 23(1):23–30, 2020. https://doi.org/10.1080/13887890.2019.1677515.

[2] H Rajabi, A Darvizeh, A Shafiei, D Taylor, and JH Dirks. Numerical investigation of insect wing fracture behaviour. *Journal of Biomechanics*, 48(1):89–94, 2015. https://doi.org/10.1016/j.jbiomech.2014.10.037.

[3] MK Salcedo and JJ Socha. Circulation in insect wings. *Integrative and Comparative Biology*, 60(5):1208–1220, 2020. https://doi.org/10.1093/icb/icaa124.

[4] H Rajabi, A Shafiei, A Darvizeh, JH Dirks, E Appel, and SN Gorb. Effect of microstructure on the mechanical and damping behaviour of dragonfly wing veins. *Royal Society Open Science*, 3(2):160006, 2016. https://doi.org/10.1098/rsos.160006.

[5] SA Combes and TL Daniel. Flexural stiffness in insect wings. II. Spatial distribution and dynamic wing bending. *Journal of Experimental Biology*, 206(17):2989–2997, 2003. https://doi.org/10.1242/jeb.00524.

[6] TE Mengesha, RR Vallance, M Barraja, and R Mittal. Parametric structural modeling of insect wings. *Bioinspiration & Biomimetics*, 4(3):036004, 2009. https://doi.org/10.1088/1748-3182/4/3/036004.

[7] D Ishihara, M Onishi, and K Sugikawa. Vein–membrane interaction in cambering of flapping insect wings. *Biomimetics*, 8(8):571, 2023. https://doi.org/10.3390/biomimetics8080571.

[8] JH Dirks and D Taylor. Veins improve fracture toughness of insect wings. *PloS One*, 7(8):e43411, 2012. https://doi.org/10.1371/journal.pone.0043411.

[9] F Haas, SN Gorb, and RJ Wootton. Elastic joints in dermapteran hind wings: materials and wing folding. *Arthropod Structure & Development*, 29(2):137–146, 2000. https://doi.org/10.1016/S1467-8039(00)00025-6.

[10] H Rajabi, A Shafiei, A Darvizeh, and SN Gorb. Resilin microjoints: a smart design strategy to avoid failure in dragonfly wings. *Scientific Reports*, 6(1):39039, 2016. https://doi.org/10.1038/srep39039.

[11] G Luo and M Sun. The effects of corrugation and wing planform on the aerodynamic force production of sweeping model insect wings. *Acta Mechanica Sinica*, 21(6):531–541, 2005. https://doi.org/10.1007/s10409-005-0072-4.

[12] T Minoda, H Nagai, S Yashiro, and N Uda. Aeroelastic effect of corrugation for an insect-sized flapping wing. *AIAA Journal*, 60(5):3180–3193, 2022. https://doi.org/10.2514/1.J061027.

[13] H Rajabi, M Rezasefat, A Darvizeh, JH Dirks, S Eshghi, A Shafiei, T Mirzababaie Mostofi, and SN Gorb. A comparative study of the effects of constructional elements on the mechanical behaviour of dragonfly wings. *Applied Physics A*, 122:1–13, 2016. http://dx.doi.org/10.1007/s00339-015-9557-6.

[14] SH Eraghi, A Toofani, RJA Guilani, S Ramezanpour, NN Bijma, A Sedaghat, A Yasamandaryaei, SN Gorb, and H Rajabi. Basal complex: a smart wing component for automatic shape morphing. *Communications Biology*, 6(1):853, 2023. https://doi.org/10.1038/s42003-023-05206-1.

[15] M Buffington and RJ Sandler. The occurrence and phylogenetic implications of wing interference patterns in Cynipoidea (Insecta: Hymenoptera). *Invertebrate Systematics*, 25(6):586–597, 2011. https://doi.org/10.1071/IS11038.

[16] A Pielowska-Ceranowska and J Szwedo. Wing interference patterns in patterned wings of *Culicoides* Latreille, 1809 (Diptera: Ceratopogonidae)-exploring potential identification tool. *Zootaxa*, 4868(3):zootaxa.4868.3.4, 2020. https://doi.org/10.11646/zootaxa.4868.3.4.

[17] A Cannet, C Simon-Chane, M Akhoundi, A Histace, O Romain, M Souchaud, P Jacob, D Sereno, K Mouline, C Barnabe, F Lardeux, P Boussès, and D Sereno. Deep learning and wing interferential patterns identify *Anopheles* species and discriminate amongst Gambiae complex species. *Scientific Reports*, 13(1):13895, 2023. https://doi.org/10.1038/s41598-023-41114-4.

[18] FS Santana, AHR Costa, FS Truzzi, FL Silva, SL Santos, TM Francoy, and AM Saraiva. A reference process for automating bee species identification based on wing images and digital image processing. *Ecological Informatics*, 24:248–260, 2014. https://doi.org/10.1016/j.ecoinf.2013.12.001.

[19] W Feindt, SJ Oppenheim, R DeSalle, PZ Goldstein, and H Hadrys. Transcriptome profiling with focus on potential key genes for wing development and evolution in *Megaloprepus caerulatus*, the damselfly species with the world's largest wings. *PLoS ONE*, 13(1):e0189898, 2018. https://doi.org/10.1371/journal.pone.0189898.

[20] J Willkommen and T Hörnschemeyer. The homology of wing base sclerites and flight muscles in Ephemeroptera and Neoptera and the morphology of the pterothorax of *Habroleptoides confusa* (Insecta: Ephemeroptera: Leptophlebiidae). *Arthropod Structure & Development*, 36 2:253–69, 2007. https://doi.org/10.1016/j.asd.2007.01.003.

[21] K Saito, H Nagai, K Suto, N Ogawa, YA Seong, T Tachi, R Niiyama, and Y Kawahara. Insect wing 3D printing. *Scientific Reports*, 11(1):18631, 2021. https://doi.org/10.1038/s41598-021-98242-y.

[22] L Mishnaevsky Jr, M Jafarpour, J Krüger, and SN Gorb. A new concept of sustainable wind turbine blades: Bio-inspired design with engineered adhesives. *Biomimetics*, 8(6):448, 2023. https://doi.org/10.3390/biomimetics8060448.

[23] A Preuss, TH Büscher, I Herzog, P Wohlsein, K Lehnert, and SN Gorb. Attachment performance of the ectoparasitic seal louse *Echinophthirius horridus*. *Communications Biology*, 7(1):36, 2024. https://doi.org/10.1038/s42003-023-05722-0.

[24] TH Büscher and SN Gorb. Convergent evolution of adhesive properties in Leaf Insect eggs and plant seeds: cross-kingdom bioinspiration. *Biomimetics*, 7(4):173, 2022. https://doi.org/10.3390/biomimetics7040173.

[25] M Jafarpour, SN Gorb, and H Rajabi. Double-spiral: a bioinspired pre-programmable compliant joint with multiple degrees of freedom. *Journal of the Royal Society Interface*, 20(198):20220757, 2023. https://doi.org/10.1098/rsif.2022.0757.

[26] H Rajabi, SH Eraghi, A Khaheshi, A Toofani, C Hunt, and RJ Wootton. An insect-inspired asymmetric hinge in a double-layer membrane. *Proceedings of the National Academy of Sciences of the USA*, 119(45):e2211861119, 2022. https://doi.org/10.1073/pnas.2211861119.

[27] G Parikh, D Rawtani, and N Khatri. Insects as an indicator for environmental pollution. *Environmental Claims Journal*, 33(2):161–181, 2021. https://doi.org/10.1080/10406026.2020.1780698.

[28] S Chowdhury, VK Dubey, S Choudhury, A Das, D Jeengar, B Sujatha, A Kumar, N Kumar, A Semwal, and V Kumar. Insects as bioindicator: A hidden gem for environmental monitoring. *Frontiers in Environmental Science*, 11:273, 2023. https://doi.org/10.3389/fenvs.2023.1146052.

[29] NR Henriques and T Cornelissen. Wing asymmetry of a butterfly community: is altitude a source of stress? *Community Ecology*, 20(3):252–257, 2019. https://doi.org/10.1556/168.2019.20.3.5.

[30] MK Asplen, G Anfora, A Biondi, DS Choi, D Chu, KM Daane, P Gibert, AP Gutierrez, KA Hoelmer, WD Hutchison, R Isaacs, ZL Jiang, Z Karpati, MT Kimura, M Pascual, CR Philips, C Plantamp, L Ponti, G Vetek, H Vogt, VM Walton, Y Yu, L Zappala, and N Desneux. Invasion biology of spotted wing Drosophila (*Drosophila suzukii*): a global perspective and future priorities. *Journal of Pest Science*, 88:469–494, 2015. https://doi.org/10.1007/s10340-015-0681-z.

[31] H Yang, MS Engel, C Shih, F Song, Y Zhao, D Ren, and T Gao. Independent wing reductions and losses among stick and leaf insects (Phasmatodea), supported by new Cretaceous fossils in amber. *BMC Biology*, 21(1):210, 2023. https://doi.org/10.1186/s12915-023-01720-0.

[32] I Deregnaucourt, J Bardin, JM Anderson, and O Bethoux. The wing venation of a new fossil species, reconstructed using geometric morphometrics, adds to the rare fossil record of Triassic Gondwanian Odonata. *Arthropod Structure & Development*, 63:101056, 2021. https://doi.org/10.1016/j.asd.2021.101056.

[33] H Rajabi, N Ghoroubi, A Darvizeh, JH Dirks, E Appel, and SN Gorb. A comparative study of the effects of vein-joints on the mechanical behaviour of insect wings: I. Single joints. *Bioinspiration & Biomimetics*, 10(5):056003, 2015. https://doi.org/10.1088/1748-3190/10/5/056003.

[34] H Rajabi, P Bazargan, A Pourbabaei, S Eshghi, A Darvizeh, SN Gorb, D Taylor, and JH Dirks. Wing cross veins: an efficient biomechanical strategy to mitigate fatigue failure of insect cuticle. *Biomechanics and Modeling in Mechanobiology*, 16:1947–1955, 2017. http://dx.doi.org/10.1007/s10237-017-0930-6.

[35] RC Herbert, PG Young, CW Smith, RJ Wootton, and KE Evans. The hind wing of the desert locust (*Schistocerca gregaria* forskål): III. a finite element analysis of a Deployable Structure. *Journal of Experimental Biology*, 203(19):2945–2955, 2000. https://doi.org/10.1242/jeb.203.19.2945.

[36] S Lin, N Chou, G Li, D Bao, Y Cai, YM Xie, and G Wang. A gradient-evolutionary coupled topology optimization for sheet reinforcement based on the mechanics of Voronoi pattern on dragonfly wings. *Advances in Engineering Software*, 190:103600, 2024. https://doi.org/10.1016/j.advengsoft.2024.103600.

[37] F Song, Y Yan, and J Sun. Energy consumption during insect flight and bioinspiration for MAV design: A review. *Computers in Biology and Medicine*, 170:108092, 2024. https://doi.org/10.1016/j.compbiomed.2024.108092.

[38] F Feng, S Xiong, Z Liu, Z Xian, Y Zhou, H Kobayashi, A Kawamoto, T Nomura, and B Zhu. Cellular topology optimization on differentiable Voronoi diagrams. *International Journal for Numerical Methods in Engineering*, 124(1):282–304, 2023. https://doi.org/10.1002/nme.7121.

[39] T Jin, NS Goo, and HC Park. Finite element modeling of a beetle wing. *Journal of Bionic Engineering*, 7:S145–S149, 2010. https://doi.org/10.1016/S1672-6529(09)60228-6.

[40] P Lin, C Wang, Y Liu, L Ren, and Z Zhang. Study of a bioinspired rigid-flexible coupling structure based on dragonfly wing by optical/electron microscopy and finite element analysis. *Micron*, 174:103534, 2023. https://doi.org/10.1016/j.micron.2023.103534.

[41] AB Kesel, U Philippi, and W Nachtigall. Biomechanical aspects of the insect wing: an analysis using the finite element method. *Computers in Biology and Medicine*, 28(4):423–437, 1998. https://doi.org/10.1016/S0010-4825(98)00018-3.

[42] RJ Wootton, RC Herbert, PG Young, and KE Evans. Approaches to the structural modelling of insect wings. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1437):1577–1587, 2003. https://doi.org/10.1098/rstb.2003.1351.

[43] MK Salcedo, J Hoffmann, S Donoughe, and L Mahadevan. Computational analysis of size, shape and structure of insect wings. *Biology Open*, 8(10):bio040774, 2019. https://doi.org/10.1242/bio.040774.

[44] SR Schachat. The wing pattern of *Moerarchis* Durrant, 1914 (Lepidoptera: Tineidae) clarifies transitions between predictive models. *Royal Society Open Science*, 4(3):161002, 2017. https://doi.org/10.1098/rsos.161002.

[45] AA Hoffmann, RE Woods, E Collins, K Wallin, A White, and JA McKenzie. Wing shape versus asymmetry as an indicator of changing environmental conditions in insects. *Australian Journal of Entomology*, 44(3):233–243, 2005. https://doi.org/10.1111/j.1440-6055.2005.00469.x.

[46] S Hardersen, SD Wratten, and CM Frampton. Does carbaryl increase fluctuating asymmetry in damselflies under field conditions? A mesocosm experiment with *Xanthocnemis zealandica* (Odonata: Zygoptera). *Journal of Applied Ecology*, 36(4):534–543, 1999. https://doi.org/10.1046/j.1365-2664.1999.00417.x.

[47] MJ Lajeunesse. Ectoparasitism of damselflies by water mites in Central Florida. *Florida Entomologist*, 90(4):643–649, 2007. https://doi.org/10.1653/0015-4040(2007)90[643:EODBWM]2.0.CO;2.

[48] SR Jongerius and D Lentink. Structural analysis of a dragonfly wing. *Experimental Mechanics*, 50:1323–1334, 2010. https://doi.org/10.1007/s11340-010-9411-x.

[49] NS Pinto, L Juen, HSR Cabette, and P De Marco. Fluctuating asymmetry and wing size of *Argia tinctipennis Selys* (Zygoptera: Coenagrionidae) in relation to riparian forest preservation status. *Neotropical Entomology*, 41:178–185, 2012. https://doi.org/10.1007/s13744-012-0029-9.

[50] A Blanke. Analysis of modularity and integration suggests evolution of dragonfly wing venation mainly in response to functional demands. *Journal of the Royal Society Interface*, 15(145):20180277, 2018. https://doi.org/10.1098/rsif.2018.0277.

[51] M Gayathri, PP Anand, and Y Shibu Vardhanan. Wing size, shape, and asymmetry analysis of the wandering glider, *Pantala flavescens* (Odonata: Libellulidae) revealed that hindwings are more asymmetric than the forewings. *Biologia*, pages 2749–2762, 2023. https://doi.org/10.1007/s11756-023-01396-5.

[52] S Dellicour, M Gerard, JG Prunier, A Dewulf, M Kuhlmann, and D Michez. Distribution and predictors of wing shape and size variability in three sister species of solitary bees. *PloS One*, 12(3):e0173109, 2017. https://doi.org/10.1371/journal.pone.0173109.

[53] CP Klingenberg, GS McIntyre, and SD Zaklan. Left–right asymmetry of fly wings and the evolution of body axes. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 265(1402):1255–1259, 1998. https://doi.org/10.1098/rspb.1998.0427.

[54] A Bonisoli-Alquati, S Ostermiller, DAE Beasley, SM Welch, AP Møller, and TA Mousseau. Faster development covaries with higher DNA damage in Grasshoppers (*Chorthippus albomarginatus*) from Chernobyl. *Physiological and Biochemical Zoology*, 91(2):776–787, 2018. https://doi.org/10.1086/696005.

[55] J Hoffmann, S Donoughe, K Li, MK Salcedo, and CH Rycroft. A simple developmental model recapitulates complex insect wing venation patterns. *Proceedings of the National Academy of Sciences of the USA*, 115(40):9905–9910, 2018. https://doi.org/10.1073/pnas.1721248115.

[56] A Tofilski. DrawWing, a program for numerical description of insect wings. *Journal of Insect Science*, 4(1):17, 2004. https://doi.org/10.1093/jis/4.1.17.

[57] AC Dobens and LL Dobens. FijiWings: an open source toolkit for semiautomated morphometric analysis of insect wings. *G3: Genes, Genomes, Genetics*, 3(8):1443–1449, 2013. https://doi.org/10.1534/g3.113.006676.

[58] S Eshghi, H Rajabi, A Darvizeh, V Nooraeefar, A Shafiei, T Mirzababaie Mostofi, and M Monsef. A simple method for geometric modelling of biological structures using image processing technique. *Scientia Iranica*, 23(5):2194–2202, 2016. https://doi.org/10.24200/sci.2016.3948.

[59] S Eshghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. WingMesh: A MATLAB-based application for finite element modeling of insect wings. *Insects*, 11(8):546, 2020. https://doi.org/10.3390/insects11080546.

[60] S Eshghi, F Nabati, S Shafaghi, V Nooraeefar, A Darvizeh, SN Gorb, and H Rajabi. An image based application in matlab for automated modelling and morphological analysis of insect wings. *Scientific Reports*, 12(1):13917, 2022. https://doi.org/10.1038/s41598-022-17859-9.

[61] CA Schneider, WS Rasband, and KW Eliceiri. NIH image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675, 2012. https://doi.org/10.1038/nmeth.2089.

[62] D Houle, J Mezey, P Galpern, and A Carter. Automated measurement of *Drosophila* wings. *BMC Evolutionary Biology*, 3(1):1–13, 2003. https://doi.org/10.1186/1471-2148-3-25.

[63] SYM Loh, Y Ogawa, S Kawana, K Tamura, and HK Lee. Semi-automated quantitative *Drosophila* wings measurements. *BMC Bioinformatics*, 18(319):1–14, 2017. https://doi.org/10.1186/s12859-017-1720-y.

[64] N Kumar, F Huizar, T Robinett, KJ Farfán-Pira, D Soundarrajan, M Unger, P Brodskiy, M Nahmad, and JJ Zartman. MAPPER: A new image analysis pipeline unmasks differential regulation of *Drosophila* wing features. *BioRxiv*, pages 1–50, 2020. https://doi.org/10.1101/2020.12.16.422888.

[65] J Lasser and E Katifori. NET: a new framework for the vectorization and examination of network data. *Source Code for Biology and Medicine*, 12(4):1–11, 2017. https://doi.org/10.1186/s13029-017-0064-3.

[66] M Dirnberger, T Kehl, and A Neumann. NEFI: network extraction from images. *Scientific Reports*, 5(1):15669, 2015. https://doi.org/10.1038/srep15669.

[67] A Cannet, C Simon-Chane, M Akhoundi, A Histace, O Romain, M Souchaud, P Jacob, P Delaunay, D Sereno, P Bousses, P Grebaut, A Geiger, CD Beer, D Kaba, and D Sereno. Wing interferential patterns (WIPs) and machine learning, a step toward automatized tsetse (*Glossina* spp.) identification. *Scientific Reports*, 12(1):20086, 2022. https://doi.org/10.1038/s41598-022-24522-w.

[68] A Cannet, C Simon-Chane, A Histace, M Akhoundi, O Romain, M Souchaud, P Jacob, D Sereno, L Gouagna, P Bousses, F Mathieu-Daude, and D Sereno. Wing interferential patterns (WIPs) and machine learning for the classification of some *Aedes* species of medical interest. *Scientific Reports*, 13(1):17628, 2023. https://doi.org/10.1038/s41598-023-44945-3.

[69] A Cannet, C Simon-Chane, A Histace, M Akhoundi, O Romain, M Souchaud, P Jacob, D Sereno, P Bousses, and D Sereno. An annotated wing interferential pattern dataset of dipteran insects of medical interest for deep learning. *Scientific Data*, 11(1):4, 2024. https://doi.org/10.1038/s41597-023-02848-y.

[70] DS Geldenhuys, S Josias, W Brink, M Makhubele, C Hui, P Landi, J Bingham, J Hargrove, and MC Hazelbag. Deep learning approaches to landmark detection in tsetse wing images. *PLoS Computational Biology*, 19(6):e1011194, 2023. https://doi.org/10.1371/journal.pcbi.1011194.

[71] DL Chopp. Some improvements of the fast marching method. *SIAM Journal on Scientific Computing*, 23(1):230–244, 2001. https://doi.org/10.1137/S106482750037617X.

[72] WB Lindquist, SM Lee, DA Coker, KW Jones, and P Spanne. Medial axis analysis of void structure in three-dimensional tomographic images of porous media. *Journal of Geophysical Research: Solid Earth*, 101(B4):8297–8310, 1996. https://doi.org/10.1029/95JB03039.

[73] H Rajabi, V Schroeter, S Eshghi, and SN Gorb. The probability of wing damage in the dragonfly *Sympetrum vulgatum* (Anisoptera: Libellulidae): a field study. *Biology Open*, 6(9):1290–1293, 2017. https://doi.org/10.1242/bio.027078.

[74] R Korkmaz, H Rajabi, S Eshghi, SN Gorb, and TH Büscher. The frequency of wing damage in a migrating butterfly. *Insect Science*, 30(5):1507–1517, 2023. https://doi.org/10.1111/1744-7917.13153.

[75] S Eshghi, M Jafarpour, A Darvizeh, SN Gorb, and H Rajabi. A simple, high-resolution, non-destructive method for determining the spatial gradient of the elastic modulus of insect cuticle. *Journal of the Royal Society Interface*, 15(145):20180312, 2018. https://doi.org/10.1098/rsif.2018.0312.

[76] M Jafarpour, S Eshghi, A Darvizeh, SN Gorb, and H Rajabi. Functional significance of graded properties of insect cuticle supported by an evolutionary analysis. *Journal of the Royal Society Interface*, 17(168):20200378, 2020. https://doi.org/10.1098/rsif.2020.0378.

[77] S Eshghi, H Rajabi, J Poser, and SN Gorb. WingSegment: A computer vision-based hybrid approach for insect wing image segmentation and 3D printing. *Advanced Intelligent Systems*, 6(5):2300712, 2024. https://doi.org/10.1002/aisy.202300712.

[78] S Eshghi, H Rajabi, S Shafaghi, F Nabati, S Nazerian, A Darvizeh, and SN Gorb. Allometric scaling reveals evolutionary constraint on Odonata wing cellularity via critical crack length. *Advanced Science*, 11(23):2400844, 2024. https://doi.org/10.1002/advs.202400844.

[79] YL Chang and X Li. Adaptive image region-growing. *IEEE Transactions on Image Processing*, 3(6):868–872, 1994. https://doi.org/10.1109/83.336259.

[80] FY Shih and S Cheng. Automatic seeded region growing for color image segmentation. *Image and Vision Computing*, 23(10):877–886, 2005. https://doi.org/10.1016/j.imavis.2005.05.015.

[81] TY Zhang and CY Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984. https://doi.org/10.1145/357994.358023.

[82] A Goyal, P Mogha, R Luthra, and N Sangwan. Path finding: A* or Dijkstra's? *International Journal in IT & Engineering*, 2(1):1–15, 2014. Google Scholar.

[83] DH Douglas and TK Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973. https://doi.org/10.3138/FM57-6770-U75U-7727.

[84] A Saalfeld. Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999. https://doi.org/10.1559/152304099782424901.

[85] DL Logan. *A first course in the finite element method*, volume 4. Thomson, Nelso, 1120 Birchomount Road, Toronto, Ontario, Canada, M1K 5G4, 2011. Google Books.

[86] H Adeli. *Supercomputing in engineering analysis*, volume 1. CRC Press, 270 Madison Avenue, New York, New York 10016, 1991. Google Books.

[87] SS Rao. *The finite element method in engineering*. Butterworth-heinemann, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, United Kingdom; 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States, 2017. Google Books.

[88] KH Huebner, DL Dewhirst, DE Smith, and TG Byrom. *The finite element method for engineers*. John Wiley & Sons, Scottsdale, Arizona, Ann Arbor, Michigan; Golden, Colorado Fort Worth, Texas, 2001. Google Books.

[89] O Panagiotopoulou. Finite element analysis (FEA): applying an engineering method to functional morphology in anthropology and human biology. *Annals of Human Biology*, 36(5):609–623, 2009. https://doi.org/10.1080/03014460903019879.

[90] ER Dumont, IR Grosse, and GJ Slater. Requirements for comparing the performance of finite element models of biological structures. *Journal of Theoretical Biology*, 256(1):96–103, 2009. https://doi.org/10.1016/j.jtbi.2008.08.017.

[91] SA Maas, BJ Ellis, GA Ateshian, and JA Weiss. FEBio: finite elements for biomechanics. *Journal of Biomechanical Engineering*, 134(1):011005, 2012. https://doi.org/10.1115/1.4005694.

[92] JA MacNeil and SK Boyd. Bone strength at the distal radius can be estimated from high-resolution peripheral quantitative computed tomography and the finite element method. *Bone*, 42(6):1203–1213, 2008. https://doi.org/10.1016/j.bone.2008.01.017.

[93] ECN Silva, MC Walters, and GH Paulino. Modeling bamboo as a functionally graded material. In *AIP Conference Proceedings*, volume 973, pages 754–759. American Institute of Physics, 2008. https://doi.org/10.1063/1.2896876.

[94] H Rajabi, M Jafarpour, A Darvizeh, JH Dirks, and SN Gorb. Stiffness distribution in insect cuticle: a continuous or a discontinuous profile? *Journal of the Royal Society Interface*, 14(132):20170310, 2017. https://doi.org/10.1098/rsif.2017.0310.

[95] A Toofani, SH Eraghi, M Khorsandi, A Khaheshi, A Darvizeh, SN Gorb, and H Rajabi. Biomechanical strategies underlying the durability of a wing-to-wing coupling mechanism. *Acta Biomaterialia*, 110:188–195, 2020. https://doi.org/10.1016/j.actbio.2020.04.036.

[96] H Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK; 40 West 20th Street, New York, NY 10011-4211, USA; 10 Stamford Road, Oakleigh, VIC 3166, Australia; Ruiz de Alarcon 13, 28014 Madrid, Spain; Dock Hous, The Waterfront, Cape Town 8001, south Africa, 2001. Google Books.

[97] L Voo, S Kumaresan, FA Pintar, N Yoganandan, and A Sances. Finite-element models of the human head. *Medical and Biological Engineering and Computing*, 34:375–381, 1996. https://doi.org/10.1007/BF02520009.

[98] M Cakmakci, G Kiziltas Sendur, and U Durak. *Guide to Simulation-Based Disciplines: Advancing Our Computational Future*, chapter Simulation-Based Engineering, pages 39–73. Springer, Gewerbestrasse 11, 6330 Cham, Switzerland, 2017. https://doi.org/10.1007/978-3-319-61264-5.

[99] PO Persson and G Strang. A simple mesh generator in MATLAB. *SIAM review*, 46(2):329–345, 2004. https://doi.org/10.1137/S0036144503429121.

[100] S Eshghi. Doctoral thesis supplementary data: "computer vision in entomology: Automated analysis and 3D modeling of insect wigns", March 2024. [Zenodo]. https://doi.org/10.5281/zenodo.10650727.

[101] L KubÍnová, J Janáček, J Albrechtová, and P Karen. *Stereological and digital methods for estimating geometrical characteristics of biological structures using confocal microscopy*. Springer Netherlands, Dordrecht, 2005. https://doi.org/10.1007/1-4020-3616-7_14.

[102] R Kienzler, H Altenbach, and I Ott. *Theories of Plates and Shells: Critical Review and New Applications*. Springer Science & Business Media, Springer-Verlag Berlin Heidelberg 2004, 2004. Google Books.

[103] H Rajabi, N Ghoroubi, K Stamm, E Appel, and SN Gorb. Dragonfly wing nodus: a one-way hinge contributing to the asymmetric wing deformation. *Acta Biomaterialia*, 60:330–338, 2017. https://doi.org/10.1016/j.actbio.2017.07.034.

[104] RJ Wootton. Functional morphology of insect wings. *Annual Review of Entomology*, 37(1):113–140, 1992. https://www.annualreviews.org/doi/pdf/10.1146/annurev.en.37.010192.000553.

[105] F Haas and RJ Wootton. Two basic mechanisms in insect wing folding. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 263(1377):1651–1658, 1996. https://doi.org/10.1098/rspb.1996.0241.

[106] K Saito, S Nomura, S Yamamoto, R Niiyama, and Y Okabe. Investigation of hindwing folding in ladybird beetles by artificial elytron transplantation and microcomputed tomography. *Proceedings of the National Academy of Sciences of the USA*, 114(22):5624–5628, 2017. https://doi.org/10.1073/pnas.1620612114.

[107] H Rajabi, JH Dirks, and SN Gorb. Insect wing damage: causes, consequences and compensatory mechanisms. *Journal of Experimental Biology*, 223(9):jeb215194, 2020. https://doi.org/10.1242/jeb.215194.

[108] SA Combes and TL Daniel. Flexural stiffness in insect wings. I. Scaling and the influence of wing venation. *Journal of Experimental Biology*, 206(17):2979–2987, 2003. https://doi.org/10.1242/jeb.00523.

[109] F Johansson, M Söderquist, and F Bokma. Insect wing shape evolution: independent effects of migratory and mate guarding flight on dragonfly wings. *Biological Journal of the Linnean Society*, 97(2):362–372, 2009. https://doi.org/10.1111/j.1095-8312.2009.01211.x.

[110] S Krishna, M Cho, H-N Wehmann, T Engels, and FO Lehmann. Wing design in flies: properties and aerodynamic function. *Insects*, 11(8):466, 2020. https://doi.org/10.3390/insects11080466.

[111] A Shahzad, FB Tian, J Young, and J Lai. Effects of hawkmoth-like flexibility on the aerodynamic performance of flapping wings with different shapes and aspect ratios. *Physics of Fluids*, 30:091902, 2018. https://doi.org/10.1063/1.5044635.

[112] H Rajabi, K Stamm, E Appel, and SN Gorb. Micro-morphological adaptations of the wing nodus to flight behaviour in four dragonfly species from the family Libellulidae (Odonata: Anisoptera). *Arthropod Structure & Development*, 47(4):442–448, 2018. https://doi.org/10.1016/j.asd.2018.01.003.

[113] CR Betts and RJ Wootton. Wing shape and flight behaviour in butterflies (Lepidoptera: Papilionoidea and Hesperioidea): a preliminary analysis. *Journal of Experimental Biology*, 138(1):271–288, 1988. https://doi.org/10.1242/jeb.138.1.271.

[114] SA Combes and TL Daniel. Into thin air: contributions of aerodynamic and inertial-elastic forces to wing bending in the hawkmoth *Manduca sexta*. *Journal of Experimental Biology*, 206(17):2999–3006, 2003. https://doi.org/10.1242/jeb.00502.

[115] J Bots, CJ Breuker, KM Kaunisto, J Koskimäki, HV Gossum, and J Suhonen. Wing shape and its influence on the outcome of territorial contests in the damselfly *Calopteryx virgo*. *Journal of Insect Science*, 12(1):96, 2012. https://doi.org/10.1673/031.012.9601.

[116] M De Block and R Stoks. Flight-related body morphology shapes mating success in a damselfly. *Animal Behaviour*, 74(4):1093–1098, 2007. https://doi.org/10.1016/j.anbehav.2007.01.023.

[117] K Berwaerts, H Van Dyck, and P Aerts. Does flight morphology relate to flight performance? An experimental test with the butterfly *Pararge aegeria*. *Functional Ecology*, 16(4):484–491, 2002. https://www.jstor.org/stable/826708.

[118] J Rudolf, LY Wang, SN Gorb, and H Rajabi. On the fracture resistance of dragonfly wings. *Journal of the Mechanical Behavior of Biomedical Materials*, 99:127–133, 2019. https://doi.org/10.1016/j.jmbbm.2019.07.009.

[119] J Schmidt, M O'Neill, JH Dirks, and D Taylor. An investigation of crack propagation in an insect wing using the theory of critical distances. *Engineering Fracture Mechanics*, 232:107052, 2020. https://doi.org/10.1016/j.engfracmech.2020.107052.

[120] WK Kim, JH Ko, HC Park, and D Byun. Effects of corrugation of the dragonfly wing on gliding performance. *Journal of Theoretical Biology*, 260(4):523–530, 2009. https://doi.org/10.1016/j.jtbi.2009.07.015.

[121] AR Ennos. The importance of torsion in the design of insect wings. *Journal of Experimental Biology*, 140(1):137–160, 1988. https://doi.org/10.1242/jeb.140.1.137.

[122] PN Sivasankaran and TA Ward. Spatial network analysis to construct simplified wing structural models for biomimetic micro air vehicles. *Aerospace Science and Technology*, 49:259–268, 2016. https://doi.org/10.1016/j.ast.2015.12.005.

[123] TY Zhang. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):337–343, 1997. https://doi.org/10.1145/357994.358023.

[124] N Aspert, D Santa-Cruz, and T Ebrahimi. MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings. IEEE International Conference on Multimedia and Expo*, volume 1, pages 705–708, Lausanne, Switzerland, 2002. IEEE. https://doi.org/10.1109/ICME.2002.1035879.

[125] P Bourke. Calculating the area and centroid of a polygon, 1988. Available Online: http://paulbourke.net/geometry/polygonmesh.

[126] SB Archibald and RA Cannings. Fossil dragonflies (Odonata: Anisoptera) from the Early Eocene Okanagan Highlands, western North America. *Canadian Entomologist*, 151(6):783–816, 2019. https://doi.org/10.4039/tce.2019.61.

[127] W Banaszak-Cibicka, M Fliszkiewicz, A Langowska, and M Żmihorski. Body size and wing asymmetry in bees along an urbanization gradient. *Apidologie*, 49:297–306, 2018. https://doi.org/10.1007/s13592-017-0554-y.

[128] S Hardersen. The role of behavioural ecology of damselflies in the use of fluctuating asymmetry as a bioindicator of water pollution. *Ecological Entomology*, 25(1):45–53, 2000. https://doi.org/10.1046/j.1365-2311.2000.00204.x.

[129] HP Yang, CS Ma, H Wen, QB Zhan, and XL Wang. A tool for developing an automatic insect identification system based on wing outlines. *Scientific Reports*, 5(1):12786, 2015. https://doi.org/10.1038/srep12786.

[130] J Hasan, A Roy, K Chatterjee, and PK Yarlagadda. Mimicking insect wings: The roadmap to bioinspiration. *ACS Biomaterials Science & Engineering*, 5(7):3139–3160, 2019. https://doi.org/10.1021/acsbiomaterials.9b00217.

[131] A Khaheshi, SN Gorb, and H Rajabi. Triple stiffness: A bioinspired strategy to combine load-bearing, durability, and impact-resistance. *Advanced Science*, 8(11):2004338, 2021. https://doi.org/10.1002/advs.202004338.

[132] A Khaheshi, SN Gorb, and H Rajabi. Spiky-joint: a bioinspired solution to combine mobility and support. *Applied Physics A*, 127(181):1–7, 2021. https://doi.org/10.1007/s00339-021-04310-5.

[133] A Khaheshi, HT Tramsen, SN Gorb, and H Rajabi. Against the wind: A load-bearing, yet durable, kite inspired by insect wings. *Materials & Design*, 198:109354, 2021. https://doi.org/10.1016/j.matdes.2020.109354.

[134] SH Eraghi, A Toofani, A Khaheshi, M Khorsandi, A Darvizeh, SN Gorb, and H Rajabi. Wing coupling in bees and wasps: From the underlying science to bioinspired engineering. *Advanced Science*, 8(16):2004383, 2021. https://doi.org/10.1002/advs.202004383.

[135] M Jafarpour, SN Gorb, and H Rajabi. Double-spirals offer the development of pre-programmable modular metastructures. *Advanced Engineering Materials*, 25(13):2300102, 2023. https://doi.org/10.1002/adem.202300102.

[136] X Bao, A Bontemps, S Grondel, and E Cattan. Design and fabrication of insect-inspired composite wings for mav application using MEMS technology. *Journal of Micromechanics and Microengineering*, 21(12):125020, 2011. https://doi.org/10.1088/0960-1317/21/12/125020.

[137] C Li, SN Gorb, and H Rajabi. Biomechanical strategies to reach a compromise between stiffness and flexibility in hind femora of desert locusts. *Acta Biomaterialia*, 134:490–498, 2021. https://doi.org/10.1016/j.actbio.2021.07.030.

[138] J Winand, SN Gorb, and TH Büscher. Gripping performance in the stick insect *Sungaya inexpectata* in dependence on the pretarsal architecture. *Journal of Comparative Physiology A*, 209(2):313–323, 2023. https://doi.org/10.1007/s00359-022-01570-1.

[139] N Chitsaz, R Marian, and J Chahl. Experimental method for 3D reconstruction of Odonata wings (methodology and dataset). *PloS One*, 15(4):e0232193, 2020. https://doi.org/10.1371/journal.pone.0232193.

[140] N Chitsaz, R Marian, A Chitsaz, and JS Chahl. Parametric and statistical study of the wing geometry of 75 species of Odonata. *Applied Sciences*, 10(15):5389, 2020. https://doi.org/10.3390/app10155389.

[141] TN Doan and CV Nguyen. A low-cost digital 3D insect scanner. *Information Processing in Agriculture*, pages 2214–3173, 2023. In press. https://doi.org/10.1016/j.inpa.2023.03.003.

[142] TJ Barrett, DJ Savage, M Ardeljan, and M Knezevic. An automated procedure for geometry creation and finite element mesh generation: Application to explicit grain structure models and machining distortion. *Computational Materials Science*, 141:269–281, 2018. https://doi.org/10.1016/j.commatsci.2017.09.048.

[143] Y Murata and Y Mitani. A study of shortest path algorithms in maze images. In *SICE Annual Conference 2011*, pages 32–33, Tokyo, Japan, 2011. IEEE. https://ieeexplore.ieee.org/abstract/document/6060570.

[144] Y Murata and Y Mitani. A fast and shorter path finding method for maze images by image processing techniques and graph theory. *Journal of Image and Graphics*, 2(1):89–93, 2014. https://doi.org/10.12720/joig.2.1.89-93.

[145] NKS Kaur. A review of various maze solving algorithms based on graph theory. *IJSRD*, 6(12):431–434, 2019. Google Scholar.

[146] O Kathe, V Turkar, A Jagtap, and G Gidaye. Maze solving robot using image processing. In *2015 IEEE Bombay Section Symposium (IBSS)*, pages 1–5, Mumbai, India, 2015. IEEE. https://ieeexplore.ieee.org/abstract/document/7456635.

[147] B Falck, D Falck, and B Collette. *Freecad [How-To]*. Packt Publishing Ltd, 35 Livery Street, Birmingham B3 2PB, UK, 2012. Google Books.

[148] J Riegel, W Mayer, and Y van Havre. Freecad. *Freecadspec2002. pdf*, 2016. https://wiki.opensourceecology.org/images/3/36/Freecadspec2002.pdf.

[149] A Okabe, B Boots, K Sugihara, and SN Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley & Sons, Baffins Lane, Chichester, West Sussex, PO19 1UD, England, 2009. https://books.google.de/books?id=dT7YH3mjeeIC.

[150] Y Takashimizu and M Iiyoshi. New parameter of roundness $R$: circularity corrected by aspect ratio. *Progress in Earth and Planetary Science*, 3(1):1–16, 2016. https://doi.org/10.1186/s40645-015-0078-x.

[151] CM Suárez-Tovar, R Guillermo-Ferreira, IA Cooper, RR Cezário, and A Córdoba-Aguilar. Dragon colors: the nature and function of Odonata (dragonfly and damselfly) coloration. *Journal of Zoology*, 317(1):1–9, 2022. https://doi.org/10.1111/jzo.12963.

[152] I Mikó, C Trietsch, T Van De Kamp, L Masner, JM Ulmer, MJ Yoder, M Zuber, EL Sandall, T Baumbach, and AR Deans. Revision of *Trassedia* (Hymenoptera: Ceraphronidae), an evolutionary relict with an unusual distribution. *Insect Systematics and Diversity*, 2(6):4, 2018. https://doi.org/10.1093/isd/ixy015.

[153] Y Fang, AD Muscente, SW Heads, B Wang, and S Xiao. The earliest elcanidae (Insecta, Orthoptera) from the Upper Triassic of North America. *Journal of Paleontology*, 92(6):1028–1034, 2018. https://doi.org/10.1017/jpa.2018.20.

[154] D Burckhardt and G Poinar. The first jumping plant-louse from mid-cretaceous burmese amber and its impact on the classification of Mesozoic psylloids (Hemiptera: Sternorrhyncha: Psylloidea s.l.). *Cretaceous Research*, 106:104240, 2020. https://doi.org/10.1016/j.cretres.2019.104240.

[155] GJ Grevera. *Distance Transform Algorithms And Their Implementation And Evaluation*, chapter Topics in Biomedical Engineering, pages 33–60. International Book Series. Springer New York, New York, NY, 2007. https://doi.org/10.1007/978-0-387-68413-0_2.

[156] EG Kokko, KD Floate, DD Colwell, and B Lee. Measurement of fluctuating asymmetry in insect wings using image analysis. *Annals of the Entomological Society of America*, 89(3):398–404, 1996. https://doi.org/10.1093/aesa/89.3.398.

[157] Z Zhihao, I Makoto, W Xishu, and L Jinsheng. The effect of bionic 3D printed structure morphology on skin friction. *Journal of Tribology*, 143(5):051103, 2021. https://doi.org/10.1115/1.4050138.

[158] D Li, B Yang, C Yang, Z Zhang, and M Hu. Effects of salt content on desiccation cracks in the clay. *Environmental Earth Sciences*, 80(671):1–13, 2021. https://doi.org/10.1007/s12665-021-09987-8.

[159] S Verykokou and C Ioannidis. An overview on image-based and scanner-based 3D modeling technologies. *Sensors*, 23(2):596, 2023. https://doi.org/10.3390/s23020596.

[160] A Perrard, M Baylac, JM Carpenter, and C Villemant. Evolution of wing shape in hornets: why is the wing venation efficient for species identification? *Journal of Evolutionary Biology*, 27(12):2665–2675, 2014. https://doi.org/10.1111/jeb.12523.

[161] T Schubnel, A Mazurier, A Nel, P Grandcolas, L Desutter-Grandcolas, F Legendre, and R Garrouste. Flat does not mean 2D: Using X-ray microtomography to study insect wings in 3D as a model for comparative studies. *Methods in Ecology and Evolution*, 14(8):2036–2048, 2023. https://doi.org/10.1111/2041-210X.14132.

[162] J Kukalova-Peck. Origin and evolution of insect wings and their relation to metamorphosis, as documented by the fossil record. *Journal of Morphology*, 156(1):53–125, 1978. https://doi.org/10.1002/jmor.1051560104.

[163] RJ Wootton. Function, homology and terminology in insect wings. *Systematic Entomology*, 4(1):81–93, 1979. https://doi.org/10.1111/j.1365-3113.1979.tb00614.x.

[164] NP Kristensen. Phylogeny of insect orders. *Annual Review of Entomology*, 26(1):135–157, 1981. https://doi.org/10.1146/annurev.en.26.010181.001031.

[165] T Schubnel, F Legendre, P Roques, R Garrouste, R Cornette, M Perreau, N Perreau, L Desutter-Grandcolas, and A Nel. Sound vs. light: wing-based communication in Carboniferous insects. *Communications Biology*, 4(1):794, 2021. https://doi.org/10.1038/s42003-021-02281-0.

[166] RR Cezário, SN Gorb, and R Guillermo-Ferreira. Camouflage by counter-brightness: the blue wings of morpho dragonflies *Zenithoptera lanei* (Anisoptera: Libellulidae) match the water background. *Journal of Zoology*, 317(2):92–100, 2022. https://doi.org/10.1111/jzo.12955.

[167] C Pélabon and TF Hansen. On the adaptive accuracy of directional asymmetry in insect wing size. *Evolution*, 62(11):2855–2867, 2008. https://doi.org/10.1111/j.1558-5646.2008.00495.x.

[168] M Santos, D Brites, and H Laayouni. Thermal evolution of pre-adult life history traits, geometric size and shape, and developmental stability in *Drosophila subobscura*. *Journal of Evolutionary Biology*, 19(6):2006–2021, 2006. https://doi.org/10.1111/j.1420-9101.2006.01139.x.

[169] ML Zelditch and DL Swiderski. Effects of procrustes superimposition and semilandmark sliding on modularity and integration: An investigation using simulations of biological data. *Evolutionary Biology*, 50(2):147–169, 2023. https://doi.org/10.1007/s11692-023-09600-9.

[170] SJM Rodrigues-Filho, CP e Castro, LF Lopes, IP da Fonseca, and MT Rebelo. Size does matter: intraspecific geometric morphometric analysis of wings of the blowfly *Chrysomya albiceps* (Diptera: Calliphoridae). *Acta Tropica*, 235:106662, 2022. https://doi.org/10.1016/j.actatropica.2022.106662.

[171] F Marini and B Walczak. Particle swarm optimization (PSO). A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015. https://doi.org/10.1016/j.chemolab.2015.08.020.

[172] R Wootton. The geometry and mechanics of insect wing deformations in flight: A modelling approach. *Insects*, 11(7):446, 2020. https://doi.org/10.3390/insects11070446.

[173] N Sontigun, KL Sukontason, BK Zajac, R Zehner, K Sukontason, A Wannasan, and J Amendt. Wing morphometrics as a tool in species identification of forensically important blow flies of Thailand. *Parasites & Vectors*, 10(229):1–14, 2017. https://doi.org/10.1186/s13071-017-2163-z.

[174] ABB Simões, RFand Wilke, CRF Chagas, RMT De Menezes, L Suesdek, LC Multini, FS Silva, MG Grech, MT Marrelli, and K Kirchgatter. Wing geometric morphometrics as a tool for the identification of *Culex* subgenus mosquitoes of *Culex* (Diptera: Culicidae). *Insects*, 11(9):567, 2020. https://doi.org/10.3390/insects11090567.

[175] ABB Wilke, RDO Christe, LC Multini, PO Vidal, R Wilk-da Silva, GC De Carvalho, and MT Marrelli. Morphometric wing characters as a tool for Mosquito identification. *PloS One*, 11(8):e0161643, 2016. https://doi.org/10.1371/journal.pone.0161643.

[176] F Shuai, S Yu, S Lek, and X Li. Habitat effects on intra-species variation in functional morphology: Evidence from freshwater fish. *Ecology and Evolution*, 8(22):10902–10913, 2018. https://doi.org/10.1002/ece3.4555.

[177] JT Haug, C Haug, and RJ Garwood. Evolution of insect wings and development–new details from Palaeozoic nymphs. *Biological Reviews*, 91(1):53–69, 2016. https://doi.org/10.1111/brv.12159.

[178] CM Clark-Hachtel, DM Linz, and Y Tomoyasu. Insights into insect wing origin provided by functional analysis of *vestigial* in the red flour beetle, *Tribolium castaneum*. *Proceedings of the National Academy of Sciences Of the USA*, 110(42):16951–16956, 2013. https://doi.org/10.1073/pnas.1304332110.

[179] J Chahl, N Chitsaz, B McIvor, T Ogunwa, JM Kok, T McIntyre, and E Abdullah. Biomimetic drones inspired by dragonflies will require a systems based approach and insights from biology. *Drones*, 5(2):24, 2021. https://doi.org/10.3390/drones5020024.

[180] R Talucdher and K Shivakumar. Tensile properties of veins of damselfly wing. *Journal of Biomaterials and Nanobiotechnology*, 4(3):247–255, 2013. http://dx.doi.org/10.4236/jbnb.2013.43031.

[181] SN Gorb, A Kesel, and J Berger. Microsculpture of the wing surface in Odonata: evidence for cuticular wax covering. *Arthropod Structure & Development*, 29(2):129–135, 2000. https://doi.org/10.1016/S1467-8039(00)00020-7.

[182] AM Mountcastle and SA Combes. Wing flexibility enhances load-lifting capacity in bumblebees. *Proceedings of the Royal Society B: Biological Sciences*, 280(1759):20130531, 2013. https://doi.org/10.1098/rspb.2013.0531.

[183] M Vanella, T Fitzgerald, S Preidikman, E Balaras, and B Balachandran. Influence of flexibility on the aerodynamic performance of a hovering wing. *Journal of Experimental Biology*, 212(1):95–105, 2009. https://doi.org/10.1242/jeb.016428.

[184] L Zhao, Q Huang, X Deng, and SP Sane. Aerodynamic effects of flexibility in flapping wings. *Journal of the Royal Society Interface*, 7(44):485–497, 2010. https://doi.org/10.1098/rsif.2009.0200.

[185] AW Shingleton, CK Mirth, and PW Bates. Developmental model of static allometry in holometabolous insects. *Proceedings of the Royal Society B: Biological Sciences*, 275(1645):1875–1885, 2008. https://doi.org/10.1098/rspb.2008.0227.

[186] DL Stern and DJ Emlen. The developmental basis for allometry in insects. *Development*, 126(6):1091–1101, 1999. https://doi.org/10.1242/dev.126.6.1091.

[187] ME Duell, C J Klok, DW Roubik, and JF Harrison. Size-dependent scaling of stingless bee flight metabolism reveals an energetic benefit to small body size. *Integrative and Comparative Biology*, 62(5):1429–1438, 2022. https://doi.org/10.1093/icb/icac131.

[188] H Rajabi and A Darvizeh. Experimental investigations of the functional morphology of dragonfly wings. *Chinese Physics B*, 22(8):088702, 2013. https://doi.org/10.1088/1674-1056/22/8/088702.

[189] XJ Li, ZH Zhang, YH Liang, LQ Ren, M Jie, and ZG Yang. Antifatigue properties of dragonfly *Pantala flavescens* wings. *Microscopy Research and Technique*, 77(5):356–362, 2014. https://doi.org/10.1002/jemt.22352.

[190] DJS Newman and RJ Wootton. An approach to the mechanics of pleating in dragonfly wings. *Journal of Experimental Biology*, 125(1):361–372, 1986. https://doi.org/10.1242/jeb.125.1.361.

[191] A Khaheshi and H Rajabi. Mechanical Intelligence (MI): A bioinspired concept for transforming engineering design. *Advanced Science*, 9(32):2203783, 2022. https://doi.org/10.1002/advs.202203783.

[192] R Sacchi and S Hardersen. Wing length allometry in Odonata: differences between families in relation to migratory behaviour. *Zoomorphology*, 132(1):23–32, 2013. https://doi.org/10.1007/s00435-012-0172-1.

[193] CM Suárez-Tovar and CE Sarmiento. Beyond the wing planform: morphological differentiation between migratory and nonmigratory dragonfly species. *Journal of Evolutionary Biology*, 29(4):690–703, 2016. https://doi.org/10.1111/jeb.12830.

[194] D Outomuro, DC Adams, and F Johansson. Wing shape allometry and aerodynamics in calopterygid damselflies: a comparative approach. *BMC Evolutionary Biology*, 13:1–11, 2013. https://doi.org/10.1186/1471-2148-13-118.

[195] S Eshghi, H Rajabi, S Shafaghi, F Nabati, S Nazerian, A Darvizeh, and SN Gorb. Supplementary material: "allometric scaling reveals evolutionary constraint on Odonata wing cellularity via critical crack length", Jan 2024. [Zenodo]. https://doi.org/10.5281/zenodo.10557201.

[196] W Zhao, Q Ma, Z Li, and C Wan. Functional compliance and protective stiffness: cross-veins in the hind wing of locust *Locusta migratoria*. *Journal of Comparative Physiology A*, 209(2):231–237, 2023. https://doi.org/10.1007/s00359-022-01587-6.

[197] R Dukas and L Dukas. Coping with nonrepairable body damage: effects of wing damage on foraging performance in bees. *Animal Behaviour*, 81(3):635–638, 2011. https://doi.org/10.1016/j.anbehav.2010.12.011.

[198] AR Ennos. Inertial and aerodynamic torques on the wings of Diptera in flight. *Journal of Experimental Biology*, 142(1):87–95, 1989. Google Scholar.

[199] N Chitsaz, K Siddiqui, R Marian, and J Chahl. An experimental study of the aerodynamics of micro corrugated wings at low Reynolds number. *Experimental Thermal and Fluid Science*, 121:110286, 2021. https://doi.org/10.1016/j.expthermflusci.2020.110286.

[200] SN Gorb. Serial elastic elements in the damselfly wing: mobile vein joints contain resilin. *Naturwissenschaften*, 86:552–555, 1999. https://doi.org/10.1007/s001140050674.

[201] S Donoughe, JD Crall, RA Merz, and SA Combes. Resilin in dragonfly and damselfly wings and its implications for wing flexibility. *Journal of Morphology*, 272(12):1409–1421, 2011. https://doi.org/10.1002/jmor.10992.

[202] R Addo-Akoto, JS Han, and JH Han. Roles of wing flexibility and kinematics in flapping wing aerodynamics. *Journal of Fluids and Structures*, 104:103317, 2021. https://doi.org/10.1016/j.jfluidstructs.2021.103317.

[203] H Fukunishi, M Hayashi, S Ito, and N Kishi. Digital photopatterning: Designing functional multipolymeric patterning films. *ACS Applied Polymer Materials*, 5(6):3888–3893, 2023. https://doi.org/10.1021/acsapm.3c00653.

[204] FG Sauer, L Jaworski, L Erdbeer, A Heitmann, J Schmidt-Chanasit, E Kiel, and R Lühken. Geometric morphometric wing analysis represents a robust tool to identify female mosquitoes (Diptera: Culicidae) in germany. *Scientific Reports*, 10(1):17613, 2020. https://doi.org/10.1038/s41598-020-72873-z.

[205] D Yin, Z Wei, Z Wang, and C Zhou. Measurement of shape and deformation of insect wing. *Review of Scientific Instruments*, 89(1):014301, 2018. https://doi.org/10.1063/1.5019200.

[206] H Shen, A Ji, Q Li, X Li, and Y Ma. Tensile mechanical properties and finite element simulation of the wings of the butterfly *Tirumala limniace*. *Journal of Comparative Physiology A*, 209(2):239–251, 2023. https://doi.org/10.1007/s00359-022-01556-z.

[207] S Jain, VD Bhatt, and S Mittal. Shape optimization of corrugated airfoils. *Computational Mechanics*, 56:917–930, 2015. https://doi.org/10.1007/s00466-015-1210-x.

[208] C Liu, P Li, F Song, and J Sun. Wing shape optimization design inspired by beetle hindwings in wind tunnel experiments. *Computers in Biology and Medicine*, 135:104642, 2021. https://doi.org/10.1016/j.compbiomed.2021.104642.

[209] HA Benítez, D Lemic, R Bažok, CM Gallardo-Araya, and KM Mikac. Evolutionary directional asymmetry and shape variation in *Diabrotica virgifera virgifera* (Coleoptera: Chrysomelidae): an example using hind wings. *Biological Journal of the Linnean Society*, 111(1):110–118, 2014. https://doi.org/10.1111/bij.12194.

[210] MS Vaca-Sánchez, P Cuevas-Reyes, I Munck, Y Oki, N Moia, T Freitas, A Almeida, K Castelan, and GW Fernandes. Patterns in wing morphology and fluctuating asymmetry in *Eulaema nigrita* along an altitudinal gradient in the brazilian rupestrian grassland. *Neotropical Entomology*, 52(5):837–847, 2023. https://doi.org/10.1007/s13744-023-01069-7.

[211] L Claußen. Studying an intrapopulational diversity in the wing asymmetry of the whitelegged damselfly *Platycnemis pennipes* (Zygoptera, Platycnemididae) using winganalogy: Insights from image analysis and computer-vison. Master's thesis, Christian-Albrechts-Universität zu Kiel, October 2023.

[212] L Kittner. Analysis of asymmetry in the wing morphology of the stick insect *Malacomorpha cyllarus*. Master's thesis, Christian-Albrechts-Universität zu Kiel, October 2023.

[213] VR de Lima, MCC de Morais, and K Kirchgatter. Integrating artificial intelligence and wing geometric morphometry to automate mosquito classification. *Acta Tropica*, 249:107089, 2023. https://doi.org/10.1016/j.actatropica.2023.107089.

[214] P Fedor, J Vaňhara, J Havel, I Malenovskỳ, and I Spellerberg. Artificial intelligence in pest insect monitoring. *Systematic Entomology*, 34(2):398–400, 2009. https://doi.org/10.1111/j.1365-3113.2008.00461.x.

[215] E Fanioudakis, M Geismar, and I Potamitis. Mosquito wingbeat analysis and classification using deep learning. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 2410–2414. IEEE, 2018. https://doi.org/10.23919/EUSIPCO.2018.8553542.

[216] A Arista-Jalife, M Nakano, Z Garcia-Nonoal, D Robles-Camarillo, H Perez-Meana, and HA Arista-Viveros. Aedes mosquito detection in its larval stage using deep neural networks. *Knowledge-Based Systems*, 189:104841, 2020. https://doi.org/10.1016/j.knosys.2019.07.012.

[217] J Park, DI Kim, B Choi, W Kang, and HW Kwon. Classification and morphological analysis of vector mosquitoes using deep convolutional neural networks. *Scientific Reports*, 10(1):1012, 2020. https://doi.org/10.1038/s41598-020-57875-1.

[218] A Gerovichev, A Sadeh, V Winter, A Bar-Massada, T Keasar, and C Keasar. High throughput data acquisition and deep learning for insect ecoinformatics. *Frontiers in Ecology and Evolution*, 9:600931, 2021. https://doi.org/10.3389/fevo.2021.600931.

[219] D Ozdemir and MS Kunduraci. Comparison of deep learning techniques for classification of the insects in order level with mobile software application. *IEEE Access*, 10:35675–35684, 2022. https://doi.org/10.1109/ACCESS.2022.3163380.

[220] TT Høye, J Ärje, K Bjerge, OLP Hansen, A Iosifidis, F Leese, HMR Mann, K Meissner, C Melvad, and J Raitoharju. Deep learning and computer vision will transform entomology. *Proceedings of the National Academy of Sciences*, 118(2):e2002545117, 2021. https://doi.org/10.1073/pnas.2002545117.

[221] K Bjerge, J Alison, M Dyrmann, CE Frigaard, HMR Mann, and TT Høye. Accurate detection and identification of insects from camera trap images with deep learning. *PLoS Sustainability and Transformation*, 2(3):e0000051, 2023. https://doi.org/10.1371/journal.pstr.0000051.

[222] S Badirli, CJ Picard, G Mohler, Z Akata, and M Dundar. Classifying the unknown: Identification of insects by deep open-set bayesian learning. *bioRxiv*, pages 2021–09, 2021. https://doi.org/10.1101/2021.09.15.460492.

[223] Z Liu, X Yan, M Qi, Y Zhu, D Huang, X Zhang, and L Lin. Artificial insect wings with biomimetic wing morphology and mechanical properties. *Bioinspiration & Biomimetics*, 12(5):056007, 2017. https://doi.org/10.1088/1748-3190/aa7f16.

[224] E Salami, PB Ganesan, Thomas A Ward, R Viyapuri, and FI Romli. Design and mechanical analysis of a 3D-printed biodegradable biomimetic micro air vehicle wing. In *IOP Conference Series: Materials Science and Engineering*, volume 152, page 012014. IOP Publishing, 2016. https://doi.org/10.1088/1757-899X/152/1/012014.

[225] M Jaffar-Bandjee, J Casas, and G Krijnen. Additive manufacturing: state of the art and potential for insect science. *Current Opinion in Insect Science*, 30:79–85, 2018. https://doi.org/10.1016/j.cois.2018.09.011.

[226] R Wu, KW Kwan, and AHW Ngan. Printed miniature robotic actuators with curvature-induced stiffness control inspired by the insect wing. *Bioinspiration & Biomimetics*, 16(4):046018, 2021. https://doi.org/10.1016/10.1088/1748-3190/abffec.

[227] HN Wehmann, L Heepe, SN Gorb, T Engels, and FO Lehmann. Local deformation and stiffness distribution in fly wings. *Biology Open*, 8(1):bio038299, 2019. https://doi.org/10.1242/bio.038299.

[228] L Yu, J Zhao, W Wang, L Zong, S Ge, and S Yan. Structural stabilization of honeybee wings based on heterogeneous stiffness. *Soft Matter*, 19(5):841–850, 2023. https://doi.org/10.1039/D2SM01353C.

# Appendices

This chapter contains supplementary materials complementing the thesis, including codes, videos, figures, and 3D models. The links and QR Codes are provided To facilitate access to these resources. These lead directly to the supplementary files on the Zenodo repository, a YouTube playlist of all relevant videos, and my website for downloading the developed software applications.

### Zenodo Repository

https://zenodo.org/doi/10.5281/zenodo.10650727

---

### YouTube Playlist

https://www.youtube.com/playlist?list=PLWsEPgdSKTKTl8Gag1X8NdlKHiyJBaQ__

---

### Download Links for Developed Applications

https://www.shahabeshghi.com/wingquest/

# Supplementary Figures

## Supplementary Figures of Introduction

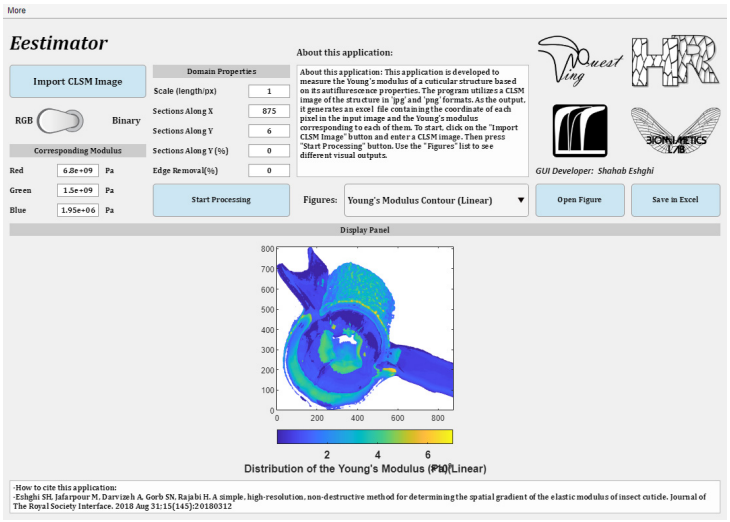



Figure S1: The graphical user interface of Eestimator- A developed software for extracting the Youngs modulus from CLSM images [75].

## Supplementary Figures of Paper II: WingGram

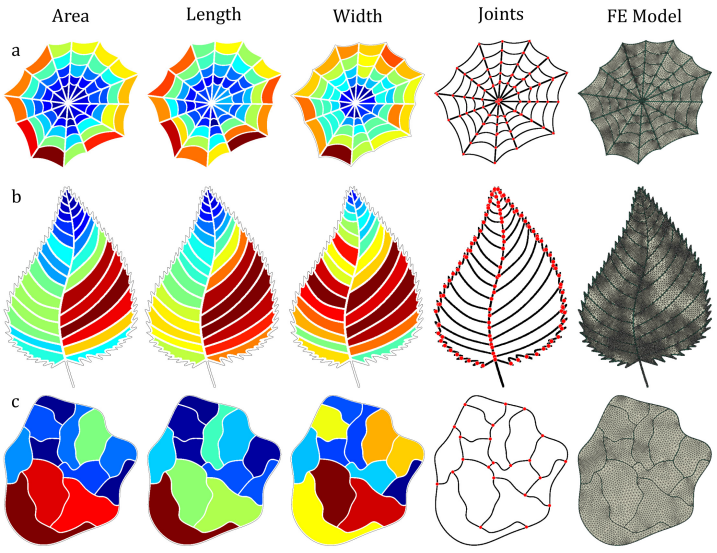



Figure S2: Extra examples showcasing WingGrams capability. a) Spider web. b) Leaf. c) Farm map.

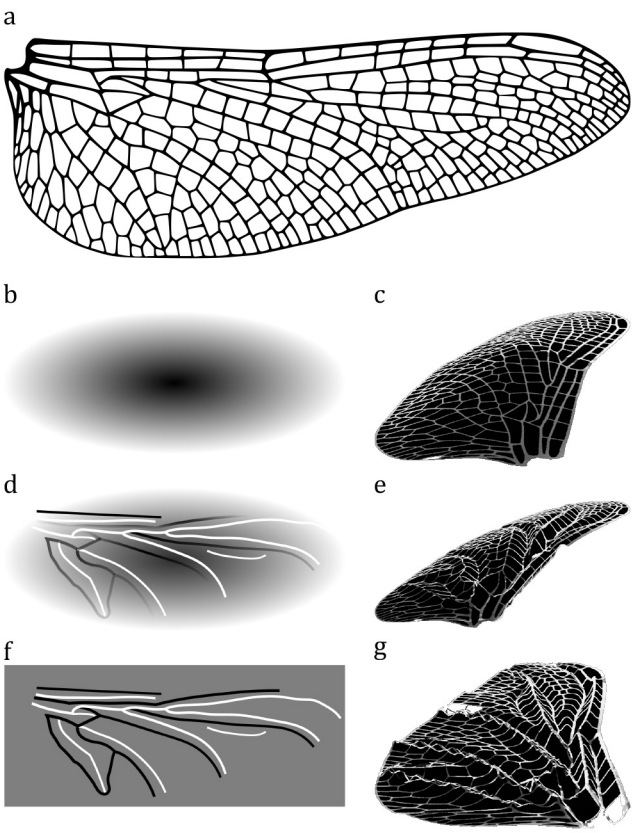



Figure S3: Assigning corrugation and concavity to the wing using WingGram. a) Image of the wing. b) Secondary image to assign concavity to the whole wing without corrugations. c) Generated model with concavity. d) Secondary image to assign both concavity and corrugations. e) Generated model with concavity and corrugations. f) Secondary image for generating model with corrugations and without concavity. g) Generated model with corrugations and without concavity.

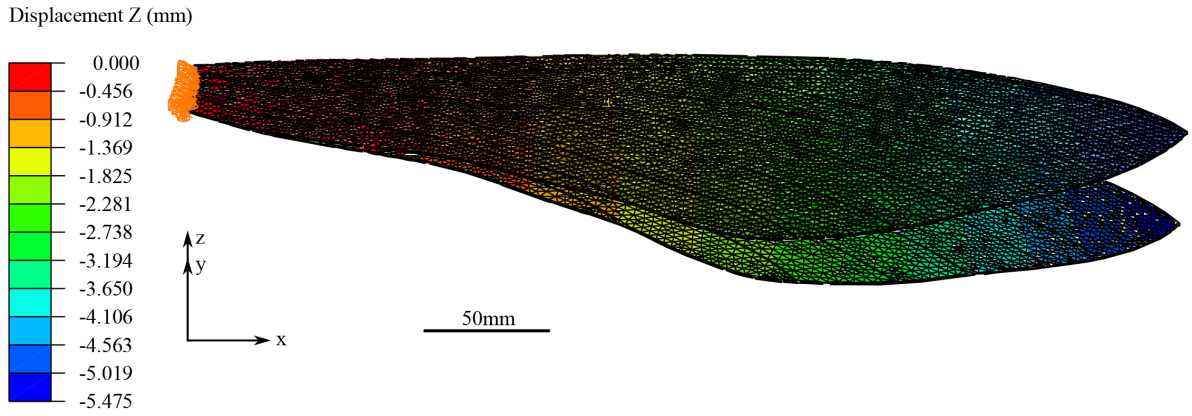



Figure S4: Finite element simulation based on *Manduca sexta* wing.

## Supplementary Figures of Paper III: WingSegment

### Validation

Figure S6 displays the forewing of the Scarlet dragonfly. After extracting the junctions and cells of this wing using WingSegment, we employed our custom Code S20 to manually count them. The corresponding supplementary files for each panel are as follows:

- Panel b (Junctions Automated Detection): *WingSegment-File S4*
- Panel c (Cells Automated Detection): *WingSegment-File S3*

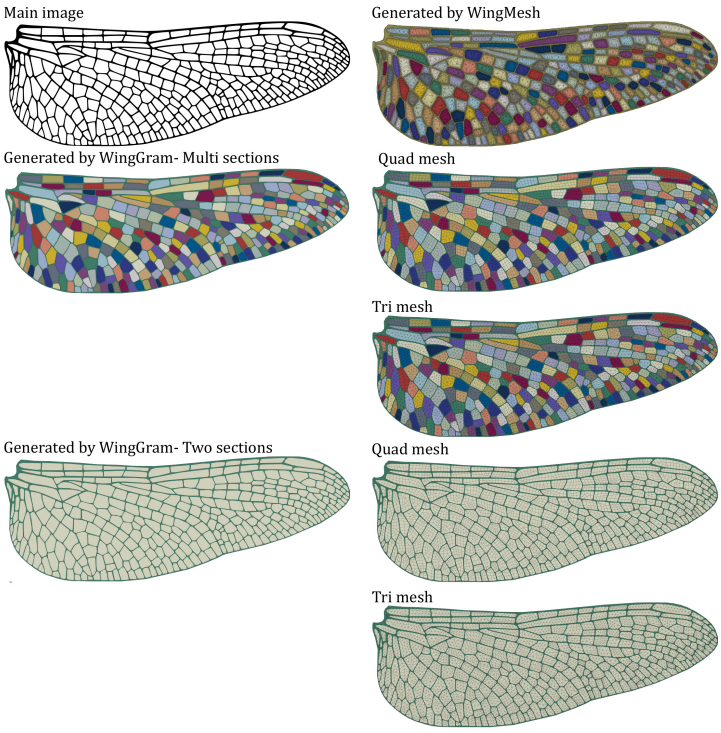



Figure S5: Comparison of WingMesh and WingGram in Generating FE model.

- Panel d (Junctions Manual Detection): *WingSegment-File S2*
- Panel e (Cells Manual Detection): *WingSegment-File S1*

Furthermore, *WingSegment Video S1* demonstrates the process of manually detecting cells, while *WingSegment Video S2* illustrates the manual detection of junctions.

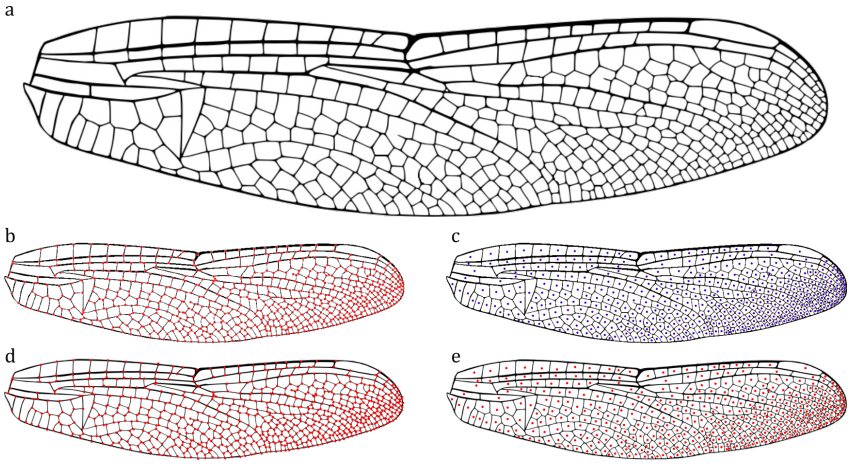



Figure S6: Manual and automated detection of cells and junctions in the forewing of a Scarlet dragonfly. a) Scarlet dragonfly forewing. b) Automated detection of junctions using WingSegment. c) Automated detection of cells using WingSegment. d) Manual detection of junctions. e) Manual detection of cells.

### Interconnection of Integrated Functions Within WingSegment

The flowchart in Figure S7 illustrates the sequence of operations in *WingSegment*. It begins with the *GetImage* function, and from there, both *RegionGrowing* and *junctionDetection* can start, with no priority between them. The *cellSegment* function also starts with *RegionGrowing*. Following *cellSegment*, the *generateFreeCadModel* function can be executed. Additionally, after running *junctionDetection*, the *veinDetection_PVD* and *veinDetection_SVD* functions can be invoked. The results obtained from *cellSegment*,

*junctionDetection*, *veinDetection_PVD*, and *veinDetection_SVD* are integrated within WingSegment, resulting in the segmentation of the wing image.

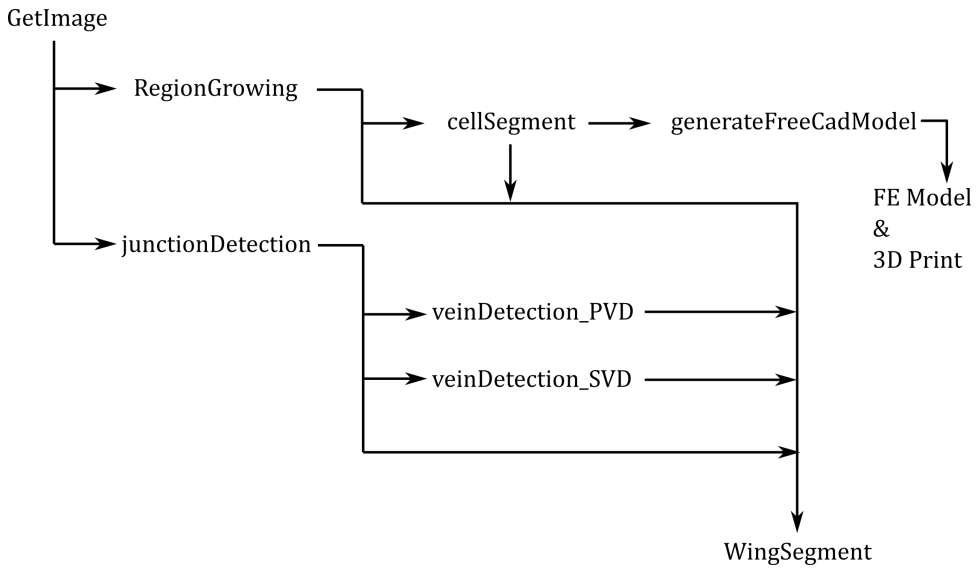



Figure S7: The flowchart illustrating the interconnection of integrated functions within WingSegment.

## Graphical User Interface of WingSegment

As mentioned, We developed a graphical user interface (GUI) for WingSegment. Figure S8 illustrates all GUI components and describes each component's performance. Figure S8 is designed in a large size. Then feel free to zoom in on each part, or you can download a pdf version of this figure via the following link:
https://zenodo.org/doi/10.5281/zenodo.8014587

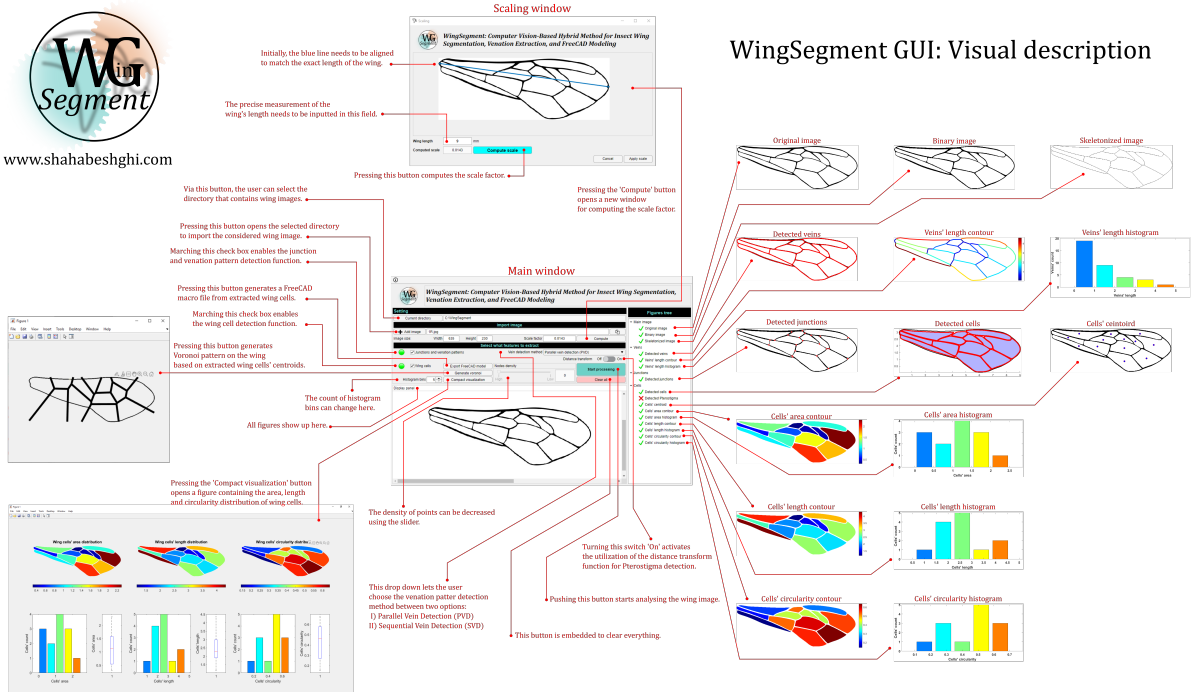



Figure S8: Description of WingSegment's graphical user interface

### Additional 3D-Printed Wings

Figure S9 illustrates extra examples showcasing the performance of WingSegment. Figure S9 provides additional examples that demonstrate the performance of WingSegment. The STL file for the wing in figure S9 is as follows:

- a, and b: *WingSegment-Model S4*.
- c, and d: *WingSegment-Model S5*.
- e, and f: *WingSegment-Model S3*.

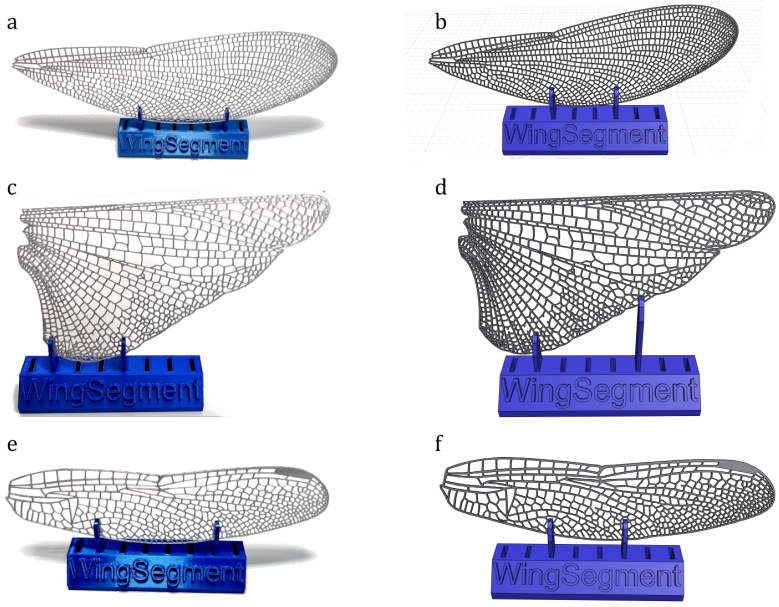



Figure S9: Additional examples of 3D-printed/model wings: a) Damselfly wing printed model, b) Damselfly wing 3D model (see also *WingAnalogy-Sample Image 4*), c) Locust wing printed model, d) Locust wing 3D model (see also *WingAnalogy-Sample Image 5*), e) Scarlet Dragonfly forewing printed model, f) Scarlet Dragonfly wing 3D model(see also *WingAnalogy-Sample Image 3*).

### Additional Quantitative Analysis of Insect Wing Morphology Using WingSegment

In this supplementary section, we present additional insights into the capabilities of WingSegment, our innovative software designed to analyze insect wing morphology comprehensively. Figures S10, S11, and S12 showcase the software's proficiency in characterizing the wings of a Damselfly, Locust, and Dragonfly, respectively. Each figure includes heatmaps illustrating wing cell area, length, and circularity, accompanied by corresponding box plots and histograms. This multi-faceted approach provides a detailed understanding of the geometric features inherent in the wings of different insect species.

Additionally, Figure S13 explores the length of venation patterns extracted from Damselfly, Locust, and Dragonfly wings in panels (a), (b), and (c), respectively. These heatmaps highlight the software's capacity to unravel the finer details of wing structures, contributing to a more comprehensive understanding of insect morphology.

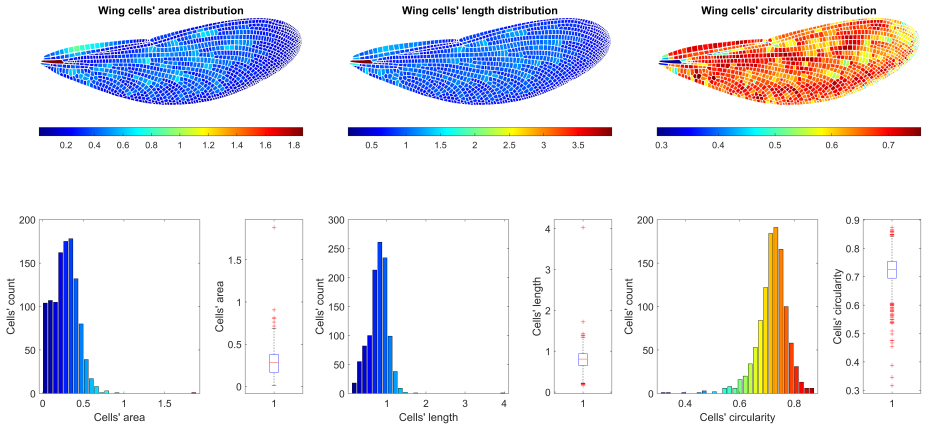



Figure S10: Morphometric Analysis of Damselfly Wing.

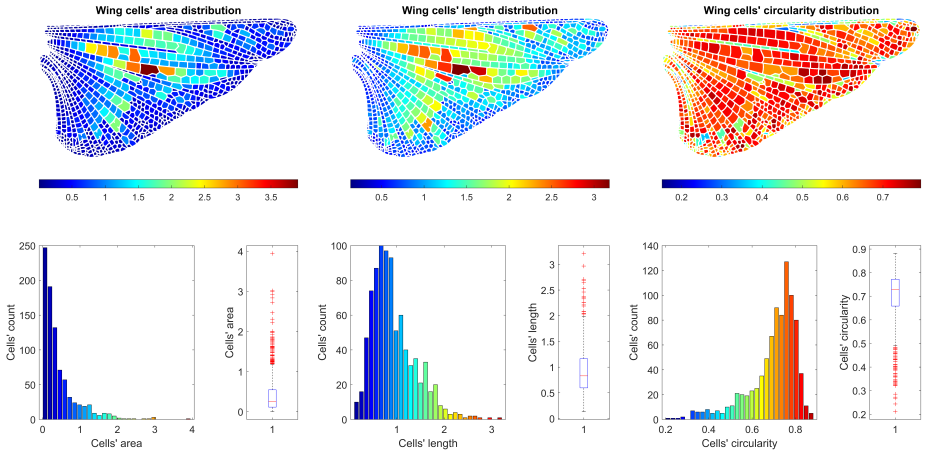



Figure S11: Exploring Locust Wing Morphology.

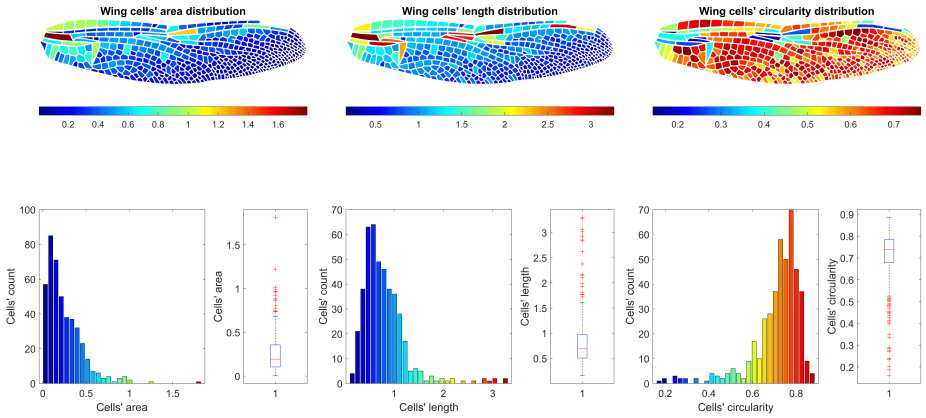



Figure S12: Dragonfly Wing Morphology Unveiled.

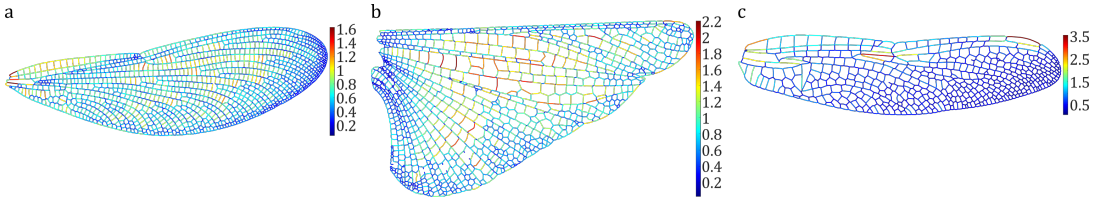



Figure S13: Panel (a), (b), and (c) present heatmaps depicting the length of venation patterns extracted from Damselfly, Locust, and Dragonfly wings, respectively.

## Supplementary Figures of Paper IV: WingAnalogy

### Graphical User Interface

WingAnalogy is a software designed for measuring the asymmetry of insect wings. Figure S14 shows the main window of WingAnalogy. Defining a new project is the first step in using the software. To initiate a new project, users should begin navigating to File > New Project, thereby defining the project properties using the *New Project* window, as demonstrated in Figure S15.

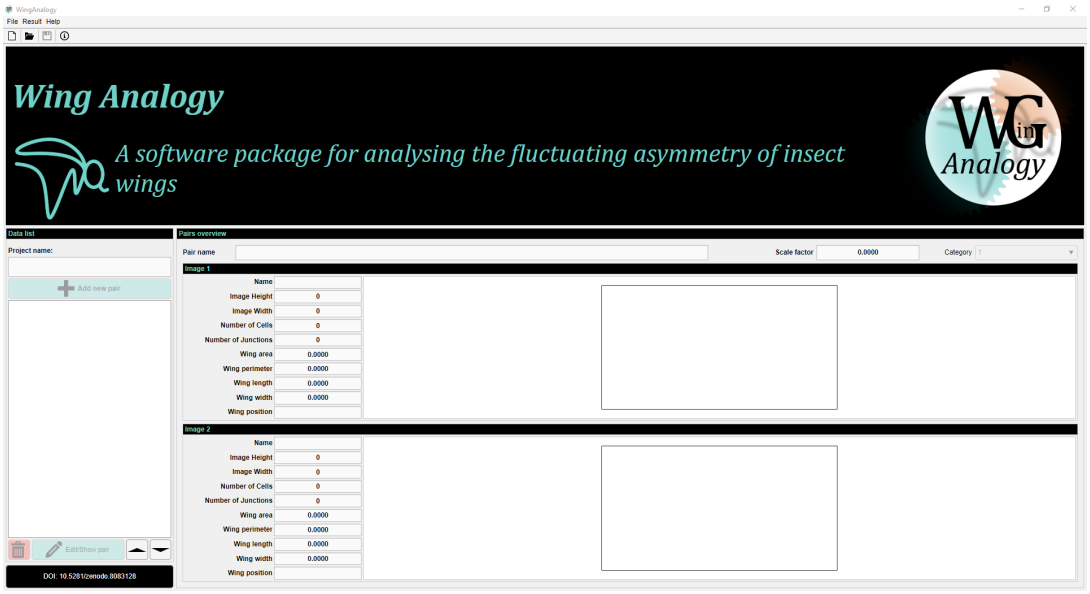



Figure S14: The main window of WingAnalogy displays a list on the left side containing the names of imported pairs. On the right side, the user can view the selected pair name, the names, and sizes of the imported images, the number of cells and junctions in each image, as well as the perimeter, length, width, and area of the wing. Additionally, the scale factor related to the pair is provided.

In the *New Project* window, as illustrated in Figure S15, the user should select a directory on their computer where all files related to the project will be stored. Additionally, for each project, the user needs to assign a name to the project, specify the project manager, and provide their email address. Furthermore, there is a text box where the user can input any important notes about the project.

After defining the new project, the user can initiate the process of measuring wing asymmetry by clicking the *Add New Pair* button. Clicking this button opens a new window, as illustrated in Figure S16. This window is named *Add/Edit Pair*. In this window, the user must import two insect wing images to be compared. For this purpose, an *Add Image* button is provided. After both images are imported, from the options

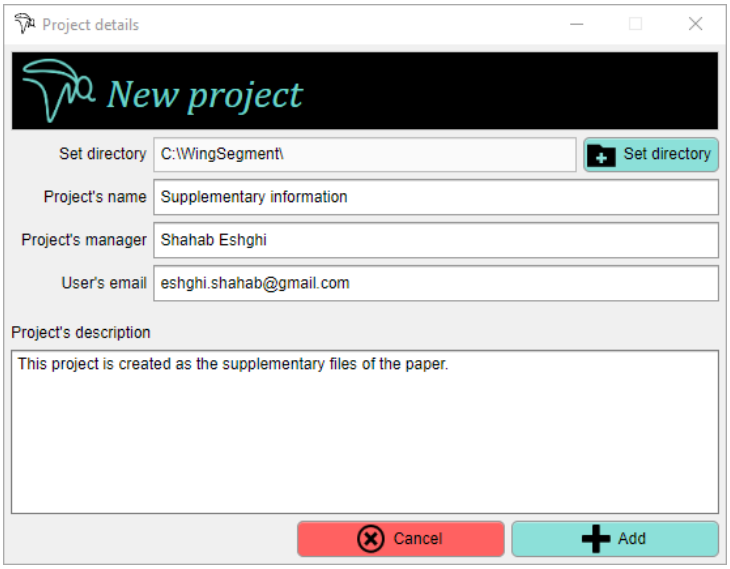



Figure S15: *New project* window. In this window, the user can define the project and provide all relevant related information.

panel on the right side in Figure S16 under the *Scaling* label, the user can scale the imported wing to its actual size. To do this, the user should adjust the line to match the corresponding length of the wing and enter the corresponding real length of the wing in the dedicated text area for this purpose.

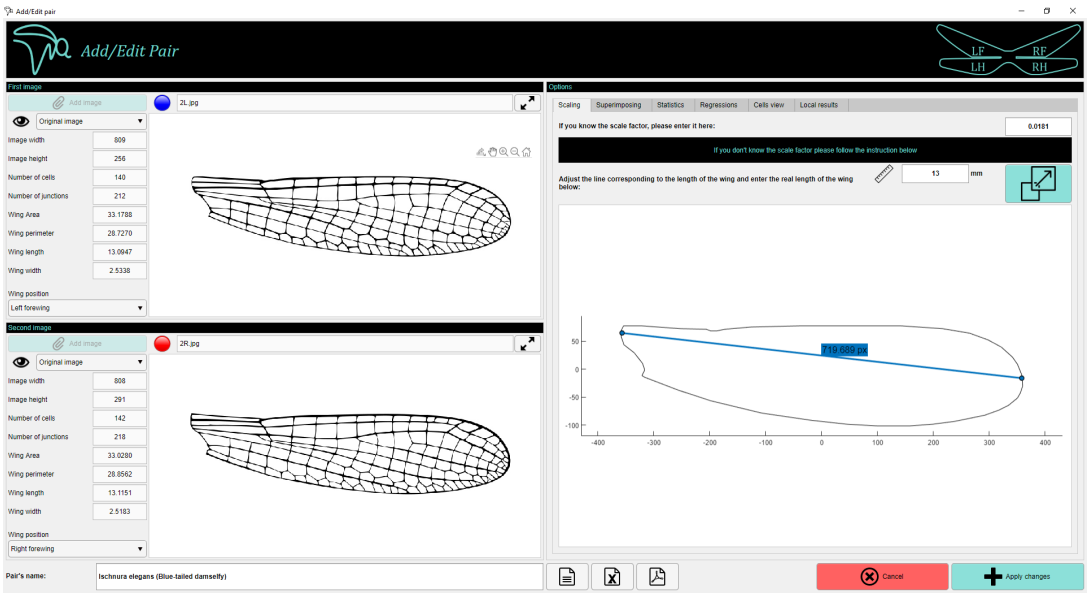



Figure S16: *Add/Edit Pair* window of WingAnalogy software

When an image is imported, the built-in algorithms segment the insect wing image and measure geometric features related to the wing and its cells. As a result, several Figures are accessible for each wing. By using the drop-down under the *Add Image* button, you can switch between different Figures such as the Binary image, the Skeletonized image, the outline of the wing, extracted cells and junctions, the histogram and distribution contour related to the cell area, length, width, and circularity which are illustrated for this example in Figure S17. Additionally, as shown in Figure S16 on the right side of the image name, a button allows you to maximize the Figure for a detailed view. Information regarding the image size, number of cells, junctions, wing area, perimeter, length, and width is displayed on the left side of the display panel for each wing. Furthermore, for

each wing, the wing position can be defined.

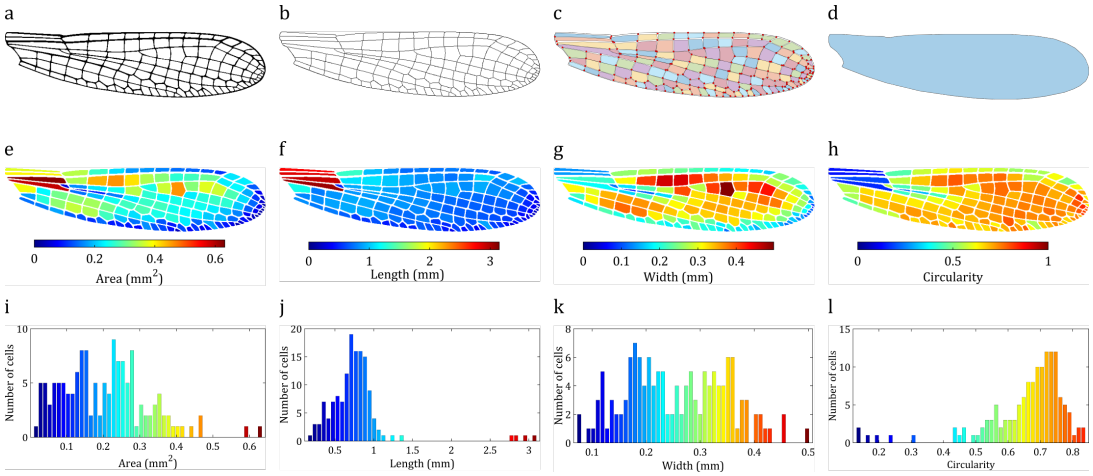



Figure S17: Generated Figures by WingAnalogy for a desired wing: a) Binary image. b) Skeletonized image. c) Extracted cells and junctions. d) Extracted wing outline. e, f, g, h) Distribution of cell area, length, width, and circularity, respectively. i, j, k, l) Histogram of cell area, length, width, and circularity, respectively.

The next crucial step is superimposing the wings. In the *Add/Edit Pair* window, this tool can be located in the options panel under the *Superimposing* label, as depicted in Figure S18. The display panel in this section presents both wings in red and blue colors, with the blue image being the reference. Users can use the arrows to reposition the red wing until it is properly superimposed. Additionally, a button labeled *Auto Superimposing* is available in this panel, which can automatically align the wings. We recommend using the auto-superimposing feature. If it does not work properly, you can try aligning the wings manually.

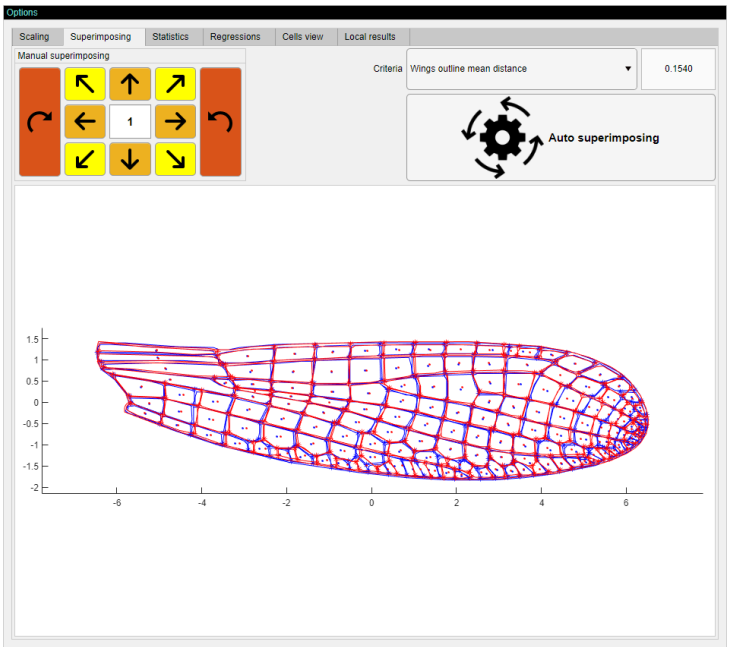



Figure S18: Superimposing tool within the *Add/Edit Pair* window. Arrow buttons are provided for manual superimposing. Additionally, there is an automated superimposing button for streamlining the process. The display panel presents both wings in different colors: blue and red.

After superimposing, as illustrated in Figure S19, you can observe the results related to the wing comparison through the *Statistics* labels in the *Options* panel. The *Statistics* label includes the NRMSE and regressions concerning the wing cell Area, Length,

Width, and Circularity. It also provides the mean distance and standard deviations of mean distances between wing outlines, junctions, and cell centroids. Additionally, the subtracted values of the junctions and cells are presented here. In the *Statistics* label, there are three columns of results labeled as *Mean Value*, *Image 1*, and *Image 2*. It is important to note that WingAnalogy considers the first image as the reference image in one scenario and the second image in another scenario. Consequently, when presenting the results, both sets of results are presented, along with the mean value calculated from them.

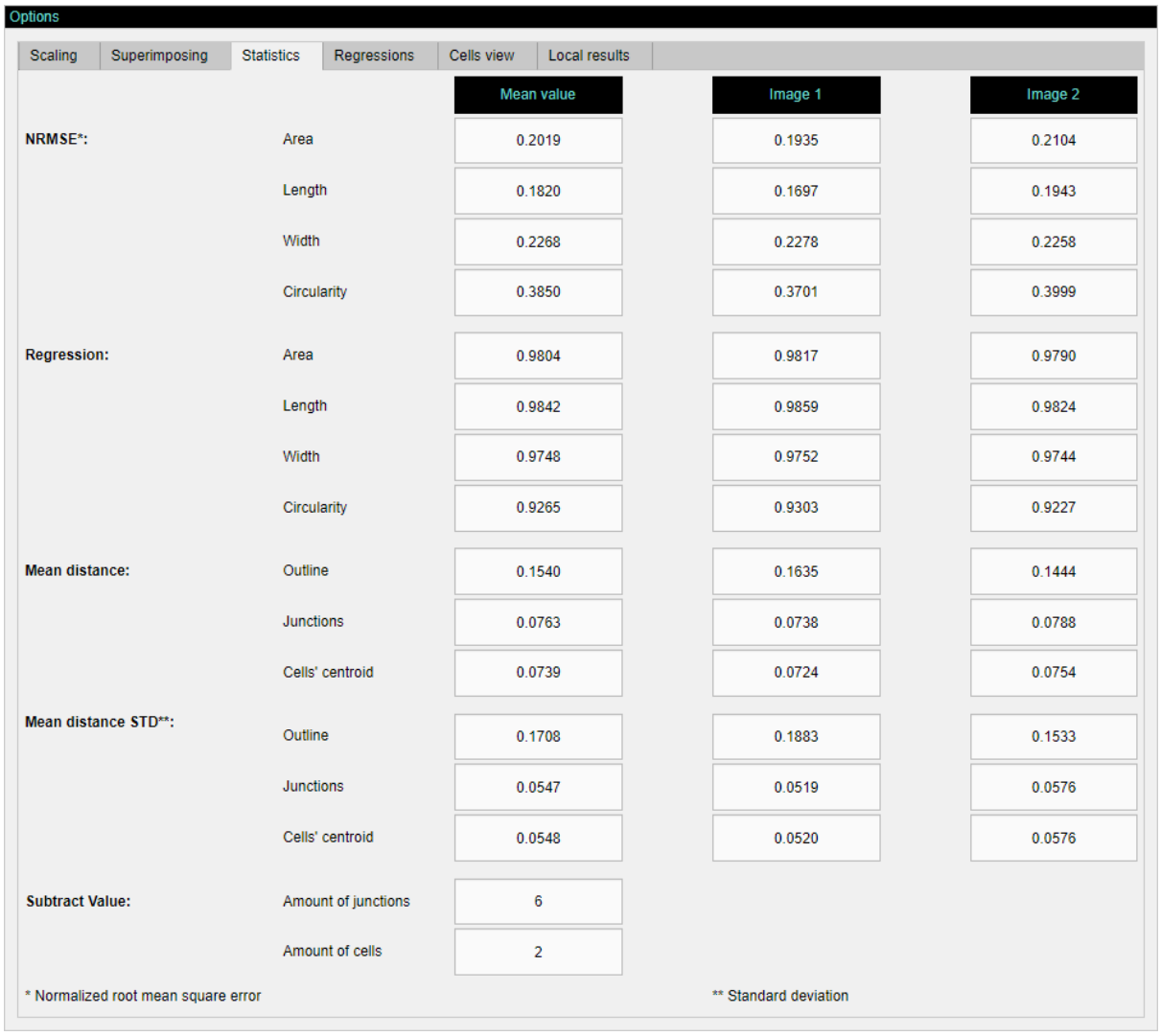



Figure S19: Statistics related to comparing wing images can be found in this label. It includes several statistics related to the NRMSE and regression of cell area, length, width, and circularity, as well as the mean distance and standard deviation between wing outlines, junctions, and cell centroids.

The next label, as shown in Figure S20, is the *Distribution Histogram*, which displays the histogram depicting the differences in the corresponding wing cell area, length, width, and circularity. Additionally, the Figures related to regressions are presented under the *Regressions* label in the Options panel, as illustrated in Figure S21.

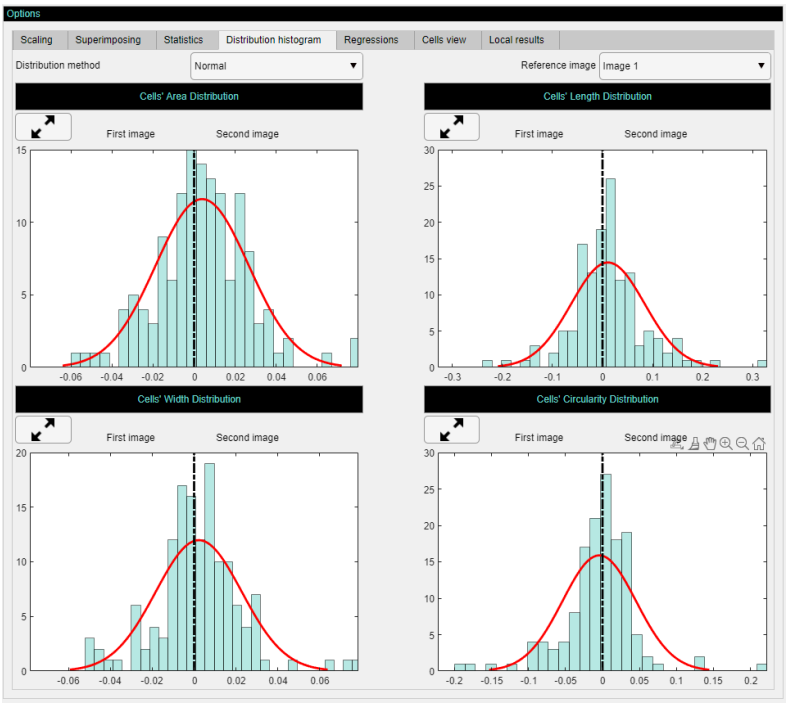



Figure S20: Histogram of cell area, length, width, and circularity with a distribution fit.

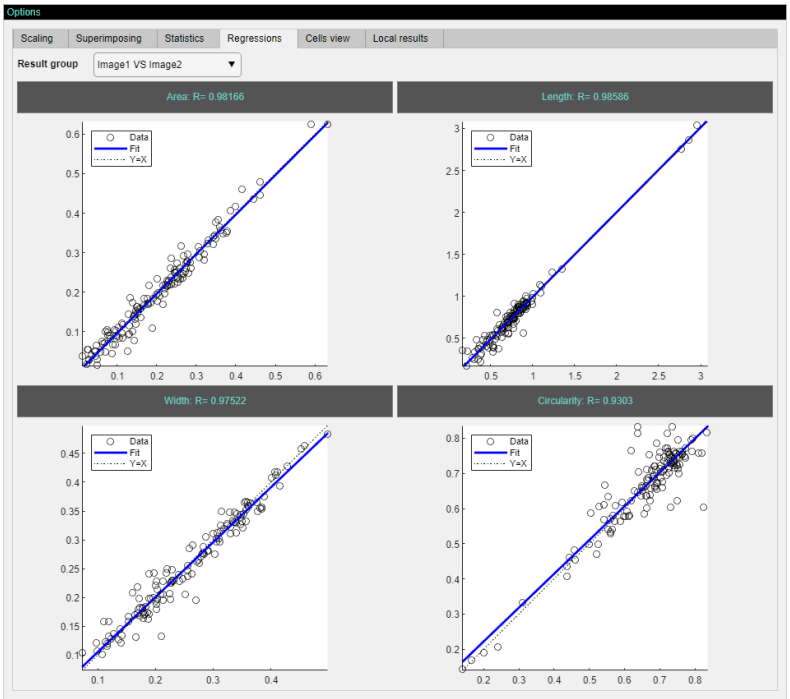



Figure S21: Graphs related to the regression of wing cell area, length, width, and circularity.

Another tool, which is illustrated in Figure S21 and embedded in the Options panel, is the *Cell View*. As mentioned before, when the user imports each image, built-in algorithms segment the wing image and measure the area, length, width, and circularity of wing cells. Consequently, in this panel, the user can view these measurements for each cell from either the first or second image separately. WingAnalogy is not only developed for comparing entire wings, but it also allows for localized comparisons. This means that

we have included a tool in which the user can divide the wing into a maximum of five different regions and obtain results for each region separately. This tool is accessible under the *Local Results* label in the Options panel. As mentioned earlier, the software can consider either the first image or the second image as the reference. Then the user can freely choose one of the images to divide the wing into several regions. Clicking the *Edit/View Sets* button opens a new window with several embedded tools, enabling the user to specify arbitrary regions within the wing and obtain results for each.

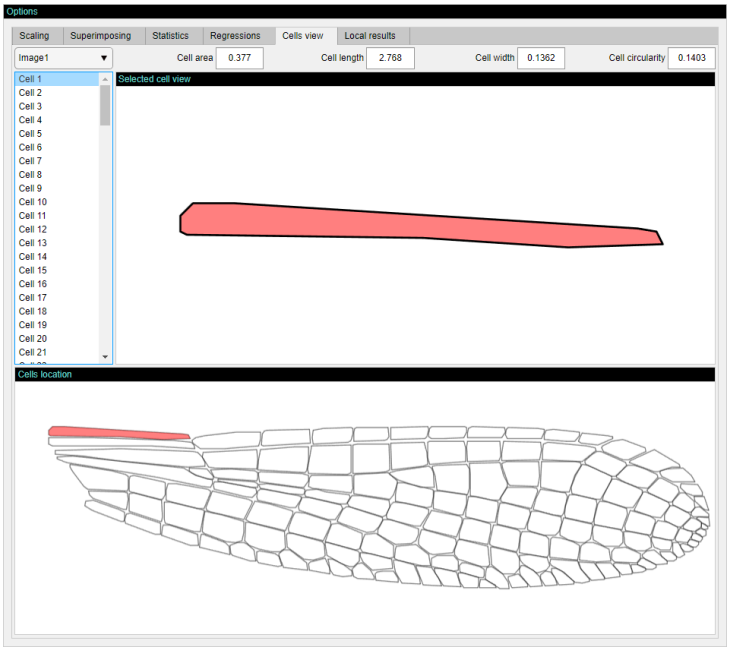



Figure S22: Cell View Tool: This tool allows the user to view the area, length, width, and circularity of individual cells. It includes a dropdown menu for changing the displayed image.

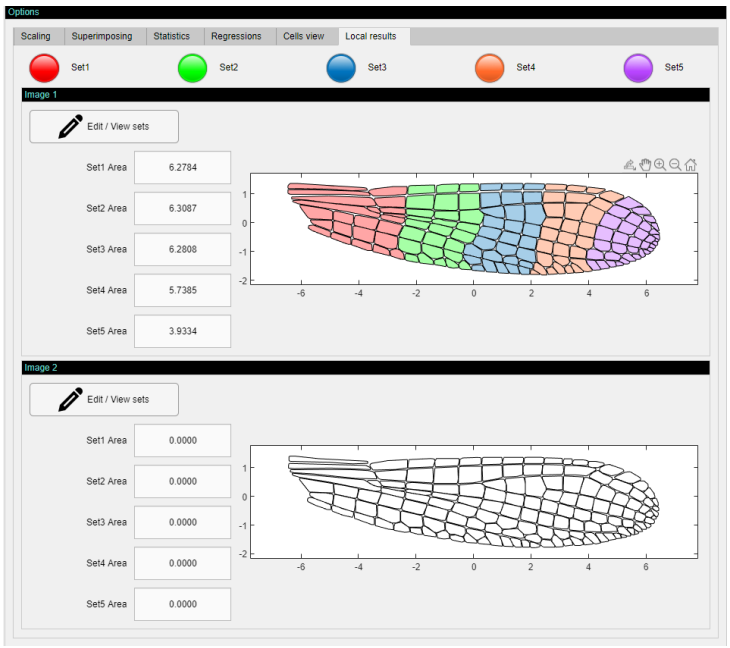



Figure S23: Under the *Local Results* label, the user can access another window, as illustrated in Figure S24, where they can define specific regions within the wing and perform localized wing comparisons.

Figure S24 displays the Local Result window. In this window, the user can access several tools for dividing the wing into a maximum of five sets. There are blue and red buttons for each set under the *Define Set* panel. The blue button activates the mouse cursor, allowing the user to outline a region on the wing by drawing a polygon around it. The centroid of each cell inside the selected region is considered part of that set. Once

a set is defined, the area of that set appears in the dedicated text box in front of it. Additionally, as soon as one set is defined, the color of that region changes to correspond with the set color, which is defined as red, green, blue, orange, and purple, respectively, for sets one to five. Furthermore, the results are displayed immediately after defining a set under the *Sets Overall Result* and *Statistics* panel.

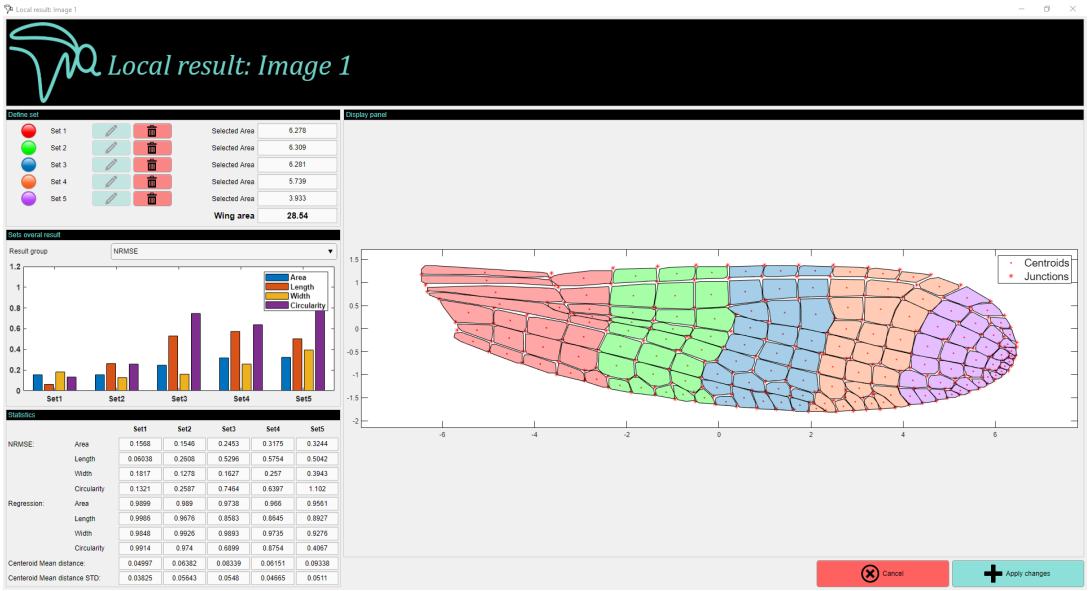



Figure S24: The Local Result window provides the user with access to tools for defining up to five sets and obtaining results from different regions.

After completing all tasks related to wing pairs, the user can utilize three embedded buttons under the *Options* panel in the *Add/Edit Pair* window, as illustrated in Figure S16, to generate a report in three different formats: TXT, PDF, or CSV files. By clicking *Apply Changes* in Figure S16, the pair is added to the list of pairs on the left side of the main window, as shown in Figure S14. Figure S25 displays the main window after four pairs have been added to the software. Clicking on the name of a pair, as shown in Figure S24, displays the images of that pair. The user can click the *Edit/Show Pair* button to return to the window shown in Figure S16 and adjust information related to the pair.

In the main window, the user can access the *All Pairs Result* window via the *Result* menu, as illustrated in Figure S26. This window compares the results of regression, NRMSE, mean distances, standard deviation of mean distances, subtract values, and wing area differences for all pairs through simple plots and box plots. All defined pairs are displayed on the left side. The numbers on the x-axis of the simple plots correspond to the pair numbers listed. The menus at the top of this window offer the user several options, including exporting Figures and their data as CSV or PDF files.

### Validation

Figure S27 displays the left and right wings of a honeybee, with labeled cells for reference. Tables S1 and S2 present the results of area, length, and circularity measurements for the right and left wings in Figure S27. These tables include measurements obtained using both WingAnalogy and ImageJ. The last three columns in both tables represent the percentage differences between the measurements from both methods. The overall findings indicate a close alignment between the two methods, providing strong evidence of the reliability of our measurement approach.

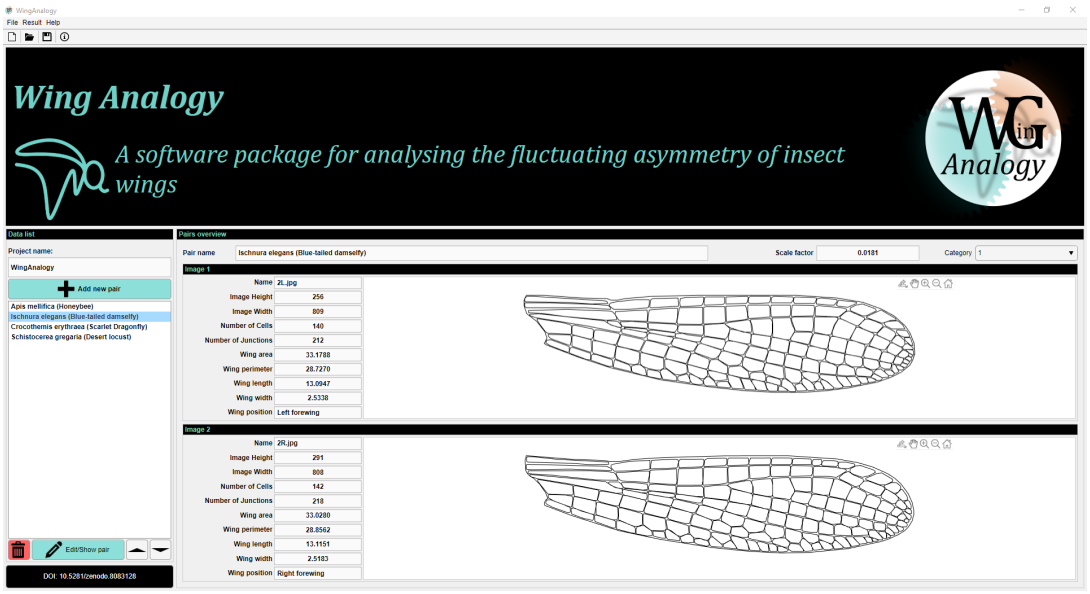



Figure S25: The main window of WingAnalogy after defining four pairs.

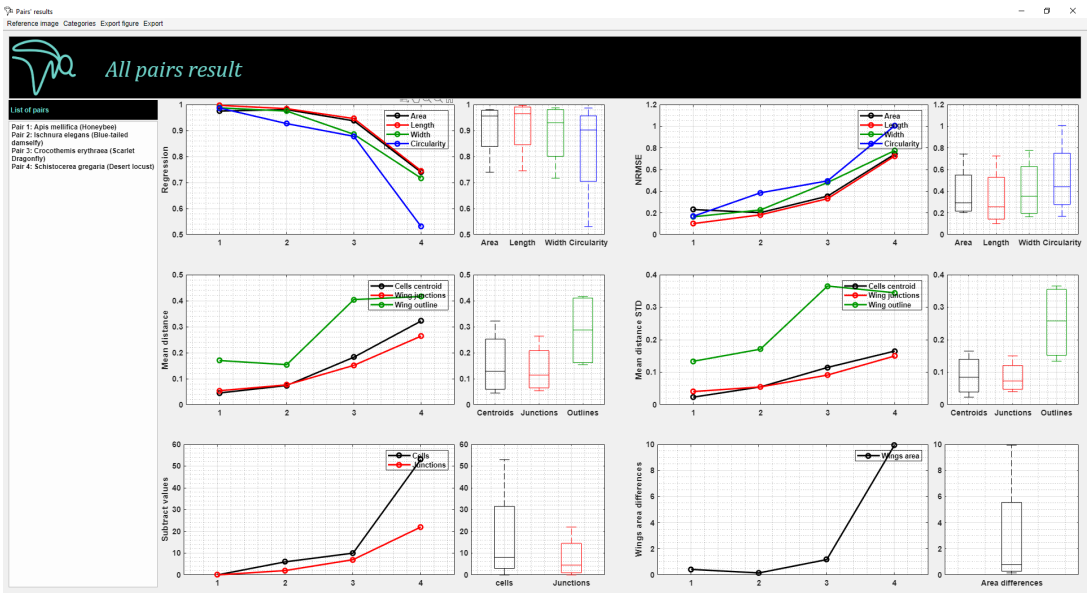



Figure S26: All pair result Window.

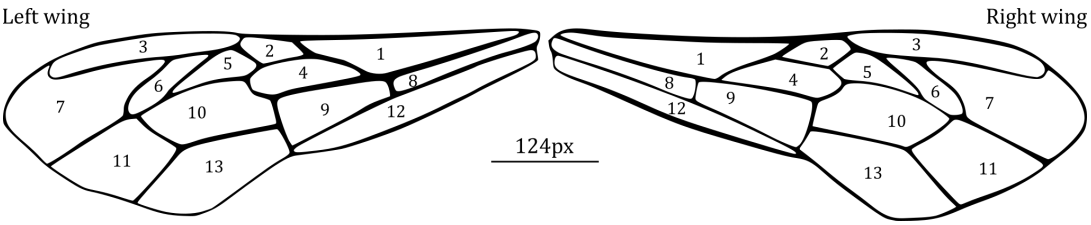



Figure S27: Visualization of the left and right wings of a honeybee with labeled cells. The labeling is intended to correspond to geometric information presented in table S4 and S5, providing insights into the characteristics of wing cells.

# Supplementary Codes

## Code S1: WingMesh

Code S1: Developed code for WingMesh

```
function boundaries=WingMesh
clear
close all
%% inputs
addpath('distmesh')
d1 = dir('*.jpg');
d2 = dir('*.tif');
d3 = dir('*.png');
str = {d1.name d2.name d3.name};
[s,v] = listdlg('PromptString','Select a file:',...
    'SelectionMode','single',...
    'ListString',str);
if v==0
    return
end
im=contrast(str{s},.9);
im=double(im)/255;
[s1,s2]=size(im);
im(1,:)=.5;
im(end,:)=.5;
im(:,1)=.5;
im(:,end)=0.5;
section_materials=[];
m=1;
n=1;
%% outer boundary
[im,border,si]=CMA(im,1,[2,2]);
s_a=size(im);
for j=1:si
    im_domain=ones(s_a(1),s_a(2));
    s2=length(border{j});
    for i=1:s2
        im_domain(border{j}(i,2),border{j}(i,1))=0;
    end
end
f=border_reorder(im_domain);
boundaries{n}=[f(:,1) f(:,2)];
section{m}=[f(:,1) f(:,2)];
final_section{m}=section;
Material1=input_material;
%% discontinuity
hole_existence=menu('Do the model have discontinuities?','Yes','No');
hole_fix=[];
if hole_existence==1
    [im,border,si]=CMA(im,0);
    hole=cell(1,si);
    for j=1:si
        im_domain=ones(s_a(1),s_a(2));
        s2=length(border{j});
       for i=1:s2
           im_domain(border{j}(i,2),border{j}(i,1))=0;
       end
        n=n+1;
        f=border_reorder(im_domain);
        boundaries{n}=[f(:,1) f(:,2)];
        hole{j}=[f(:,1) f(:,2)];
        hole_fix=[hole_fix;f(:,1) f(:,2)];
    end
    imshow(im);
else
    hole={};
end
fin = menu('Continue marking subdomains?','Yes','No');
%%
figure(1)
```

```
while fin==1
    m=m+1;
    hold on
    [im,border,si]=CMA(im,1);
    section=cell(1,si);
    for i=1:si
        im_domain=ones(s_a(1),s_a(2));
        s2=length(border{i});
       for j=1:s2
           im_domain(border{i}(j,2),border{i}(j,1))=0;
       end
        n=n+1;
        f=border_reorder(im_domain);
        boundaries{n}=[f(:,1) f(:,2)];
        section{i}=[f(:,1) f(:,2)];
    end
    final_section{m}=section;
    Material=input_material;
    section_materials=[section_materials;Material];
    imshow(im);
    fin = menu('Continue marking subdomains?','Yes','No');
end
section_materials=[section_materials;Material1];
clf
close all
pv=[];
for i=2:length(boundaries)
    pv=[pv;boundaries{i}];
end
 pv=[pv;hole_fix];
[p,t]=distmesh2d(@fd1,@fh1,1,[0,0;s2,s1],pv,50,boundaries);
clf
save all_structure.mat
[sections,p]=Domain_separator(p,t,final_section,hole,boundaries);
INP(sections,p,section_materials)
end
function wl=contrast(w,c_r)
w=imread(w);
if numel(size(w))==3
    w=rgb2gray(w);
end
c=255*c_r;
s=size(w);
wl=reshape(w,s(1)*s(2),1);
wl(wl>c)=255;
wl(wl~=255)=0;
wl=reshape(wl,s(1),s(2));
end
function [sections,node]=Domain_separator(p,t,final_section,hole,boundaries)
s_section=length(final_section);
col=hsv(s_section+1);
element=t;
node=p;
% corrugations
s_h=length(node);
node_round=round(node);
Z=zeros(s_h,1);
is_3D=menu('3D? ','Yes','No');
if is_3D==1
    w_h=height;
    w_h=w_h(:,end:-1:1);
    w_h=w_h*100;
    for i=1:s_h
        Z(i)=w_h(node_round(i,2),node_round(i,1));
    end
else
    for i=1:s_h
        Z(i)=0;
    end
end
node=[node Z];
middle=(node(t(:,1),1:2)+node(t(:,2),1:2)+node(t(:,3),1:2))/3;
s_hole=length(hole);
```

```
id_hole=[];
for j=1:s_hole
    inside=inpolygon(middle(:,1),middle(:,2),hole{j}(:,1),hole{j}(:,2));
    id=find(inside==1);
    id_hole=[id_hole; id];
end
sections=cell(1,s_section);
middle(id_hole,:)=[];
element(id_hole,:)=[];
for i=2:s_section
    id_all=[];
    s_cell=length(final_section{i});
    final_element=[];
    for j=1:s_cell
        inside=inpolygon(middle(:,1),middle(:,2),final_section{i}{j}(:,1),final_section{
            i}{j}(:,2));
        id=find(inside==1);
        final_element=[final_element;element(id,:)];
        id_all=[id_all; id];
    end
    sections{i-1}=final_element;
    patch('vertices',node,'faces',final_element,'edgecol','k','facecol',col(i,:));
    drawnow
    axis equal
    element(id_all,:)=[];
    middle(id_all,:)=[];
end
if ~isempty(element)
    sections{s_section}=element;
end
patch('vertices',node,'faces',element,'edgecol','k','facecol',col(1,:));
drawnow
hold on
for i=1:length(boundaries)
   plot(boundaries{i}(:,1),boundaries{i}(:,2),'g')
end
end
function border=border_reorder(im)
[id1,id2]=find(im==0);
in_y=id1(1);  in_x=id2(1);
first_y=in_y;
first_x=in_x;
border=[];
ss=size(im);
ask=menu('Show border ordering','Yes','No');
if ask==1
    fig=figure(2);
    imshow(im(end:-1:1,:))
    hold on
end
i=1;
stop=0;
while ~stop
    if i==2
        im(first_y,first_y)=0;
    end
    w_n=[im(in_y-1,in_x) im(in_y,in_x+1) im(in_y+1,in_x) im(in_y,in_x-1)];
    id=find(w_n==0);
    out=[in_x,in_y-1;in_x+1,in_y;in_x,in_y+1;in_x-1,in_y];
    if isempty(id)
        w_n=[im(in_y-1,in_x+1) im(in_y+1,in_x+1) im(in_y+1,in_x-1) im(in_y-1,in_x-1)];
        id=find(w_n==0);
        out=[in_x+1 in_y-1;in_x+1 in_y+1;in_x-1 in_y+1;in_x-1 in_y-1];
    end
    if ask==1
        plot(in_x,ss(1)-in_y+1,'o')
        axis([0 ss(2) 0 ss(1)])
       m(i)=getframe(fig);
    end
    in_y=out(id(1),2);  in_x=out(id(1),1);
    im(in_y,in_x)=255;
    border=[border;in_x in_y];
    i=i+1;
```

```
    if in_y==first_y && in_x==first_x
       stop=1;
    end
end
border=[border; border(1,:)];
ask_smooth=menu('Smooth Edge?','YES','NO');
switch ask_smooth
    case 1
        border=smooth_edge(border,100);
    case 2
        border=smooth_edge(border,2);
end
end
function [im,border,si]=CMA(im,remove_border,varargin)            %Cack Mould Algorithm
imshow(im)
s_image=size(im);
if isempty(varargin)
    [x,y]=getpts;
else
    x=varargin{1}(1);
    y=varargin{1}(2);
end
si=size(x,1);
for j=1:si
    x1=round(x(j)); y1=round(y(j));
    FWP=[x1,y1];       % FWP:found white pixel(first white pixel is choosen by user)
    border_domain=[];
    while ~isempty(FWP)
        X_plus=[FWP(:,1)    ;  FWP(:,1)+1   ;   FWP(:,1)     ; FWP(:,1)-1];
        Y_plus=[FWP(:,2)-1  ;   FWP(:,2)    ;  FWP(:,2)+1   ;   FWP(:,2)];
        coord=[X_plus Y_plus];
        coord=unique(coord,'rows');
        l=size(coord,1);
        new_FWP=[];
        new_FBP=[];
        for i=1:l
            if im(coord(i,2),coord(i,1))==1
                im(coord(i,2),coord(i,1))=0.8;
                new_FWP=[new_FWP; coord(i,1) coord(i,2)];   %new_FWP: new Found White
                    Pixel
            elseif im(coord(i,2),coord(i,1))==0
                im(coord(i,2),coord(i,1))=0.5;
                new_FBP=[new_FBP; coord(i,1) coord(i,2)];    %new_FBP: new Found Black
                    Pixel
            end
        end
        X_cross=[FWP(:,1)-1; FWP(:,1)-1 ;FWP(:,1)+1 ;FWP(:,1)+1];
        Y_cross=[FWP(:,2)-1 ;FWP(:,2)+1 ;FWP(:,2)-1 ;FWP(:,2)+1];
        coord=[X_cross Y_cross];
        coord=unique(coord,'rows');
        l=size(coord,1);
        for i=1:l
            if im(coord(i,2),coord(i,1))==0
                im(coord(i,2),coord(i,1))=0.5;
                new_FBP=[new_FBP;coord(i,1) coord(i,2)];    %new_FBP: new Found Black
                    Pixel
            end
        end
        border_domain=[border_domain;new_FBP];
        FWP=new_FWP;
    end
    if remove_border
        s=length(border_domain);
        for i=1:s
            im(border_domain(i,2),border_domain(i,1))=0;
        end
    end
    border_domain(:,2)=s_image(1)-border_domain(:,2);
    border{j}=border_domain;
end
imshow(im)
end
function d=fh1(p,s)
```

```
a=.5;
b=.25;
c=5;
for k=1:size(s,2)
    pv=s{k};
    for i=1:size(p,1)
        dd(i,k)=dpoly(p(i,:),pv);
    end
end
for i=1:size(p,1)
    d(i)=min([min(abs(dd(i,:))*b+a) c]) ;
end
d=d';
mean(d)
end
function d=fd1(p,boundaries)
pv=boundaries{1};
d=zeros(size(p,1),1);
for i=1:size(p,1)
    d(i)=dpoly(p(i,:),pv);
end
end
function INP(sections,node,Material)
s1=size(sections);
s2=size(node,1);
h=waitbar(1/7,'Heading');
fid_inp=fopen('abaqus_output.inp','w');
fprintf(fid_inp,'*Heading \n');
fprintf(fid_inp,'**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ \n');
fprintf(fid_inp,'**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ \n');
fprintf(fid_inp,'** Generated by: Modified distmesh2d \n');
fprintf(fid_inp,'** Eshghi et al., "Modified distmesh2d: a hybrid digital image
    processing and mesh generation method in MATLAB" \n');
fprintf(fid_inp,'** For more information please contact: Dr. Hamed Rajabi(harajabi@
    hotmail.com)\n');
fprintf(fid_inp,'** Website: http://www.biomimeticslab.ir\n');
fprintf(fid_inp,['** developed on: ' date '\n']);
fprintf(fid_inp,'**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n');
waitbar(2/7,h,'Nodes');
fprintf(fid_inp,'*Node\n');
node=[(1:size(node,1))' node];
all_elements=[];
for i=1:s1(2)
    all_elements=[all_elements;sections{i}];
end
all_elements=all_elements(:);
for i=1:s2
    id= all_elements==node(i,1);
    if sum(id)~=0
        fprintf(fid_inp,'      %g,%g,%g,%g\n',node(i,1),node(i,2),node(i,3),node(i,4));
    end
end
fprintf(fid_inp,'*Element, type=S3R\n');
waitbar(3/7,h,'Detecting Elments');
count=1;
ss=0;
for i=1:s1(2)
    ss=[ss;length(sections{i})];
    for j=1:ss(end)
        fprintf(fid_inp, '%g,%g,%g,%g\n',count,sections{i}(j,1),sections{i}(j,2),
            sections{i}(j,3));
        count=count+1;
    end
end
a=0;
b=0;
eee=[];
for i=1:s1(2)
    fprintf(fid_inp,'\n *Elset, elset=section%g, internal \n',i);
    a=a+1;
    b=b+ss(i+1);
    eee=[eee;a b];
    for j=a:b
```

```
351             fprintf(fid_inp,'%g,',j);
352             if rem(j,16)==0
353                 fprintf(fid_inp,'\n');
354             end
355         end
356         a=b; b=a;
357     end
358     fprintf(fid_inp,'\n**MATERIALS \n');
359     waitbar(6/7,h,'Materials');
360     for i=1:s1(2)
361         fprintf(fid_inp,'*Material, name=Material-%g \n',i);
362         fprintf(fid_inp,'*Density \n');
363         fprintf(fid_inp,'%g \n',Material(i,1));
364         G=sqrt(Material(i,2)*Material(i,3))/(2*(1+Material(i,4)));
365         fprintf(fid_inp,'*Elastic,type=LAMINA\n');
366         fprintf(fid_inp,'%g,%g,%g,%g,%g,%g \n',Material(i,2),Material(i,3),Material(i,4),G,G
                ,G);
367     end
368     waitbar(7/7,h,'Ending');
369     for i=1:s1(2)
370         fprintf(fid_inp,'**Section:section-%g\n',i);
371         fprintf(fid_inp,'*shell section, elset=section%g, material=Material-%g\n',i,i);
372     end
373     close(h); fclose('all');
374     end
375     function Fix_points=smooth_edge(f,IT)
376     x=f(:,1);
377     y=f(:,2);
378     l=length(x);
379     for j=1:IT
380         x(1)=sum([x(end-1) 20*x(1) x(2)])/22;
381         x(end)=x(1);
382         y(1)=sum([y(end-1) 20*y(1) y(2)])/22;
383         y(end)=y(1);
384         for i=2:l-1
385             x(i)=sum([x(i-1) 20*x(i) x(i+1)])/22;
386             y(i)=sum([y(i-1) 20*y(i) y(i+1)])/22;
387         end
388     end
389     Fix_points=[x y];
390     end
391     function Material=input_material
392     prompt = {'Density:','E1:','E2:',' Poissons Ratio'};
393     dlg_title = 'Section Material';
394     defaultanswer={'1200','1e6','2e6','0.3'};
395     num_lines = 1;
396     answer = inputdlg(prompt,dlg_title,num_lines,defaultanswer);
397     Material=[str2double(answer{1}) str2double(answer{2}) str2double(answer{3}) str2double(
            answer{4})];
398     end
399     function w=height
400     d1 = dir('*.jpg');
401     d2 = dir('*.tif');
402     d3 = dir('*.png');
403     str = {d1.name d2.name d3.name};
404     [s,v] = listdlg('PromptString','Supplementary Image',...
405         'SelectionMode','single',...
406         'ListString',str);
407     if v==0
408         return
409     end
410     w=imread(str{s});
411     w=rgb2gray(w);
412     w=(double(w)/255);
413     u=4:size(w,2)-4;
414     u1=4:size(w,1)-4;
415     for i1=1:200
416         w(u1,u)=(10*w(u1,u)+w(u1,u+1)+w(u1,u+2)+w(u1,u+3)+w(u1,u-1)+w(u1,u-2)+w(u1,u-3)...
417             +w(u1-1,u)+w(u1-2,u)+w(u1-3,u)+w(u1+1,u)+w(u1+2,u)+w(u1+3,u))/22;
418     end
419     end
```

## Code S2: Get Image

Code S2 is the function developed for importing the wing image. Line 1 in Code S2 defines the function.

**Input:** The input for this function is not explicitly defined. To import the wing image, we utilized the *uigetfile* function in MATLAB, allowing the user to import a desired file (in this case, the wing image). The output of *uigetfile* is the 'file' name and the 'path' where the file is stored.
**Ouput:** The outputs of this function is as follows:

- *imageBinaryImage*: This variable stores the binarized image of the wing.
- *imageOrgImage*: This variable stores the original matrix of the imported wing image.
- *imageWidth*: This variable stores the width of the imported image.
- *imageHeight*: This variable stores the height of the imported image.

**Code Description:** This code reads the image matrix in line 5. Line 6 checks whether the image is in RGB format. If the image is in RGB, the *rgb2gray* function in line 7 converts the image to grayscale. Line 9 is included to binarize the image using the *imbinarize* function. This function is a built-in MATLAB function that utilizes Otsu's method. The function requires a threshold value, and we have set it to 0.54 in this code. Lines 10 to 14 handle the cropping of the image to fit within a frame.

Code S2: GetImage code in MATLAB, for manual importing of the wing Image

```
1  function [imageBinaryImage,imageOrgImage,imageWidth,imageHeight]=GetImage
2  [file,path,indx] = uigetfile('*.*');
3  ThresholdEditField = 0.54;
4  if indx
5      imageOrgImage = imread([path file]);
6      if size(imageOrgImage,3)==3
7          imageOrgImage = rgb2gray(imageOrgImage);
8      end
9      imageBinaryImage= imbinarize(imageOrgImage,ThresholdEditField); %Otsu's Method
10     [r,c] = find(imageBinaryImage==0);
11     imageBinaryImage = imageBinaryImage(min(min(r))-3:max(max(r))+3,min(min(c))-3:max(
           max(c))+3);
12    imageOrgImage = imageOrgImage(min(min(r))-3:max(max(r))+3,min(min(c))-3:max(max(c))
          +3);
13     imageWidth = size(imageOrgImage,2);
14     imageHeight = size(imageOrgImage,1);
15 end
16 end
```

## Code S3: Region Growing

Region growing, as introduced in the main manuscript, extracts the boundary of a closed domain surrounded by black pixels.
**Inputs:** Three inputs are defined for Code S3:

- *imageBinaryImage*: The binary image, the first output of Code S2.
- $r$: The row number (y-coordinate) of a random seed inside a desired region.
- $c$: The column number (x-coordinate) of a random seed inside a desired region.

**Outputs:** Code S3 has four outputs:

- *cell_whiteID*: An N-by-2 array storing the coordinates of detected white pixels inside the desired region, used to generate the corresponding matrix for area, length, and circularity distribution of wing cells.
- *n_white*: The number of detected white pixels in the desired region, representing the area of the detected region.
- *im*: The image after detection of the region. When a desired region in the wing is detected, its color changes to gray to avoid repeated detection.
- *boundary*: An N-by-2 array containing the coordinates of the detected region.

**Code Description:**

- **line 3:** The binary image's matrix is converted to double because, upon importing an image into the function, the color of the detected area changes to gray, which is not possible in a binary image.
- **Line 4:** *FWP* represents 'Found White Pixels'. Initially, $r$ and $c$ (the second and third inputs of Code S3) go into *FWP*. This variable updates during the algorithm's execution.
- **Lines 5-6:** *border* and *cell_whiteID* are empty matrices that store the coordinates of detected boundary pixels and white pixels inside the region, respectively.
- **Lines 7-34:** The while loop starts from the initial seed point ($r$, $c$) and checks for new white (lines 8-18) and black pixels (lines 19-30) around the seed pixel. Each time it detects a new white or black pixel, it saves their coordinates separately. Also, each time it finds white pixels, it changes the color of those detected pixels and continues checking for new white pixels around those detected points. The while loop continues until there are still white pixels inside the region.
- **Lines 35-59:** In the while loop, detected pixels on the boundary are not in order. It means that the coordinates don't define a closed loop, and they are only points on the boundary that are stored irregularly. In lines 35-59, the code constructs a closed loop of those extracted points on the boundary of the domain.

To demonstrate how to run this function and use it for another project, we have developed Code S9. In this code, the user can simply import a desired image and click inside one region, and the code extracts that region's boundary.

Code S3: Region Growing Code in MATLAB

```
1  function [cell_whiteID,n_white,im,boundary]= RegionGrowing(imageBinaryImage,r,c)
2  %r, and c represent the row and column number of a random seed inside the desired region
3  im = double(imageBinaryImage);
4  FWP=[c,r];       % FWP:found white pixel(first white pixel is choosen by user)
5  border=[];
6  cell_whiteID=[];
7  while ~isempty(FWP) %the code continues while it does not find new white pixe
8      X=[FWP(:,1) FWP(:,1)     FWP(:,1)+1     FWP(:,1)     FWP(:,1)-1];
9      Y=[FWP(:,2) FWP(:,2)-1     FWP(:,2)     FWP(:,2)+1     FWP(:,2)];
10     X=X(:); Y=Y(:);
11     l=length(X);
12     new_FWP=[];
13     for i=1:l
14         if im(Y(i),X(i))==1
15             im(Y(i),X(i))=0.8; % changes the color of found white pixel to bright grey
```

```
16                  new_FWP=[new_FWP; X(i) Y(i)];    %#ok %new_FWP: new Found White Pixel
17              end
18          end
19          % Finding new black pixels in the neighbor of FWP on 8 directions
20          X=[FWP(:,1) FWP(:,1)+1 FWP(:,1) FWP(:,1)-1 FWP(:,1)-1 FWP(:,1)-1 FWP(:,1)+1 FWP(:,1)
                +1];
21          Y=[FWP(:,2)-1 FWP(:,2) FWP(:,2)+1 FWP(:,2) FWP(:,2)-1 FWP(:,2)+1 FWP(:,2)-1 FWP(:,2)
                +1];
22          X=X(:); Y=Y(:);
23          l=length(X);
24          new_FBP=[];
25          for i=1:l
26              if im(Y(i),X(i))==0
27                  im(Y(i),X(i))=0.1; % changes the color of found black pixel to dark grey
28                  new_FBP=[new_FBP; X(i) Y(i)];    %#ok %new_FBP: new Found Black Pixel
29              end
30          end
31          border=[border;new_FBP];                                %#ok
32          cell_whiteID=[cell_whiteID;new_FWP];     %#ok
33          FWP=new_FWP;
34      end
35      border = unique(border,'rows');
36      n_white=size(unique(cell_whiteID,'rows'),1);
37      im_bw = zeros(size(im,1),size(im,2));
38      for i=1:size(border,1)
39          im_bw(border(i,2),border(i,1))=1;
40          im(border(i,2),border(i,1))=0;
41      end
42      BW = imbinarize(im_bw);
43      try
44          boundary = bwtraceboundary(BW, [border(i,2),border(i,1)],'S');
45      catch
46          try
47              boundary = bwtraceboundary(BW, [border(i,2),border(i,1)],'E');
48          catch
49              try
50                  boundary = bwtraceboundary(BW, [border(i,2),border(i,1)],'N');
51              catch
52                  boundary = bwtraceboundary(BW, [border(i,2),border(i,1)],'W');
53              end
54          end
55      end
56      boundary = [boundary(:,2) boundary(:,1)];
57      boundary(:,2)=size(im,1)-boundary(:,2);
58      [boundary(:,1), boundary(:,2)] = poly2cw(boundary(:,1), boundary(:,2));
59      im = double(im);
60      end
```

## Code S4: Wing Cell Segmentation

This code is developed to segment all cells of a wing by repeatedly applying the *Region_Growing* function.
**Input:** The only input of this function is the binary image generated by the *GetImage* function.
**Outputs:** Code S4 has six outputs as follows:

- *ContourMatrix_Area*: Stores the matrix generated by *RegionGrowing* that contains the corresponding matrix for the distribution of wing cells' areas.
- *ContourMatrix_Length*: Similar to the previous one, this variable contains the distribution of wing cells' lengths.
- *ContourMatrix_Circularity*: This variable similarly contains the distribution of wing cells' circularities.

- *wingOutline*: This is an N-by-2 array that contains the coordinates of the wing outline, extracted from *RegionGrowing*.
- *wingCells*: This is a one-by-N cell array. The number of cells in this array represents the number of wing cells, and each cell array contains the coordinates of one wing cell boundary.
- *wingInfo*: This is an N-by-6 matrix. N represents the number of cells. Each row of this variable has six columns representing the cell index, the x-coordinate of the cell centroid, the y-coordinate of the cell centroid, the cell's area, the cell's length, and the cell's circularity.

**Code Description:**

- **Lines 8-11:** These lines generate a gray frame around the image. This is necessary for extracting the outline of the wing. Otherwise, the *RegionGrowing* function faces problems when it reaches the edge of the image (first and last columns and rows of the image matrix).
- **Line 16:** *RegionGrowing* runs in this line to extract the outline of the wing.
- **Lines 32-63:** In line 32, a *while* loop starts. In this while loop, the *RegionGrowing* function runs recursively until all cells are detected. As mentioned, the second and third inputs of the *RegionGrowing* function are one seed point inside the region. Keep in mind that the outline is already detected, and as a result, the color of the outer region of the wing image is already gray. To find the first seed inside the first cell, in line 30 the code finds all white pixels, and in line 33, it uses only the first detected white pixel. As the outer region of the wing is gray, this white pixel is located inside one of the cells. Then, *RegionGrowing* extracts that specific cell's boundary and changes the color of that cell to gray. Then inside the loop in line 60, the code again searches for new white pixels, and absolutely these white pixels are inside not detected cells because all detected cells have changed color to gray. This while loop continues until the code finds no new white pixels. *WingSegment Video S3* in the supplementary materials visually demonstrates this procedure.

Code S4: Wing Cells Segmentation Code in MATLAB

```
1  function [ContourMatrix_Area,...
2      ContourMatrix_Length,...
3      ContourMatrix_Circularity,...
4      wingOutline,...
5      wingCells,...
6      wingInfo] = cellSegment(imageBinaryImage)
7  w = double(imageBinaryImage);
8  w(1,:)=.5;
9  w(end,:)=.5;
10 w(:,1)=.5;
11 w(:,end)=0.5;
12 s_im=size(w);
13 r=2; c=2;
14 wingInfo = [];
15 wingCells ={};
16 [cell_whiteID,~,w,boundary]=RegionGrowing(w,c,r);
17 wingOutline= boundary;
18 ContourMatrix_Area=zeros(s_im(1),s_im(2));
19 for i=1:size(cell_whiteID,1)
20     ContourMatrix_Area(cell_whiteID(i,2),cell_whiteID(i,1))=NaN;
21 end
22 ContourMatrix_Length=zeros(s_im(1),s_im(2));
23 for i=1:size(cell_whiteID,1)
```

```
    ContourMatrix_Length(cell_whiteID(i,2),cell_whiteID(i,1))=NaN;
end
ContourMatrix_Circularity=zeros(s_im(1),s_im(2));
for i=1:size(cell_whiteID,1)
    ContourMatrix_Circularity(cell_whiteID(i,2),cell_whiteID(i,1))=NaN;
end
[r,c]=find(w==1);
count=0;
allDetected = 0;
while sum(r)>0    [cell_whiteID,n_white,w,boundary]=RegionGrowing(w,r(1),c(1));
    if n_white > 8
        wingCells{end+1}= boundary;
        count=count+1;
        s_b=size(boundary,1);
        all_l=[];
        for i=1:s_b
            l=sqrt(((boundary(i,1)-boundary(:,1)).^2)+(boundary(i,2)-boundary(:,2)).^2);
            all_l=[all_l;l];  %#ok
        end
        all_ll = [];
        for i=1:size(boundary,1)-1
            ll=sqrt(((boundary(i,1)-boundary(i+1,1)).^2)+(boundary(i,2)-boundary(i+1,2))
                .^2);
            all_ll=[all_ll;ll];  %#ok
        end
        Perimeter = sum(all_ll);
        Rc = Perimeter/(2*pi);
        for i=1:size(cell_whiteID,1)
            if ~isempty(all_l)
                ContourMatrix_Area(cell_whiteID(i,2),cell_whiteID(i,1))=n_white;
                ContourMatrix_Length(cell_whiteID(i,2),cell_whiteID(i,1))=max(all_l);
                ContourMatrix_Circularity(cell_whiteID(i,2),cell_whiteID(i,1)) = n_white
                    /(pi*(Rc^2));
            end
        end
        wingInfo(count,:) = [count...
        mean(boundary(:,1))...
        mean(boundary(:,2))...
        n_white...
        max(all_l)...
        n_white/(pi*(Rc^2))];
    end
    [r,c]=find(w==1);
    id_gray = w==0.1;
    w(id_gray) = 0;
end
id = ContourMatrix_Area == 0;
ContourMatrix_Area(id) = nan;
ContourMatrix_Area = ContourMatrix_Area(end:-1:1,:);
id= ContourMatrix_Length == 0;
ContourMatrix_Length(id) = nan;
ContourMatrix_Length = ContourMatrix_Length(end:-1:1,:);
id= ContourMatrix_Circularity == 0;
ContourMatrix_Circularity(id) = nan;
ContourMatrix_Circularity = ContourMatrix_Circularity(end:-1:1,:);
end
```

## Code S5: Junction Detection/Skeletonization

This function is developed to detect junctions of the wing.
**Input:** The only input for this function is the binary image of the wing.
**Outputs:** This function has two outputs:

- *junctions*: A structure variable with the following fields:
  - *junctions.c*: An N-by-1 array storing the X coordinate of junctions.
  - *junctions.r*: An N-by-1 array storing the Y coordinate of junctions.

  - *junctions.surround.x*: An N-by-20 array containing the X coordinate of a circle with a radius of 1.5 pixels around junctions (used in the detection of veins).
  - *junctions.surround.y*: An N-by-20 array containing the Y coordinate of a circle with a radius of 1.5 pixels around junctions (used in the detection of veins).

  N in this array represents the number of detected junctions.

- *skeletonizedImage*: A matrix representing the skeletonized image of the wing.

**Code Description:**

- **Lines 5-11:** These lines apply the distance transform on the wing image. This feature can be turned on or off by setting the value of the *Distancetransform* variable to 1 or 0. Distance transform can detect the location of the Pterostigma in the wing.
- **Line 14:** Applies the thinning method to generate the skeletonized image.
- **Lines 17-40:** Implements the conditions used to detect the locations of junctions.
- **Lines 44-52:** Modifies detected junctions to remove junctions located within a distance of less than 3 pixels from each other.

Code S5: Junction Detection code in MATLAB, for automated Wing Junction Detection

```
1  function [junctions, skeletonizedImage]=junctionDetection(imageBinaryImage)
2  wOrg = imageBinaryImage;
3  w = imageBinaryImage;
4  Distancetransform = 0; %This value can be 0 or 1
5  if Distancetransform
6      w = bwdist(w);
7      w = w/max(max(w));
8      w(w>0.7) =1;
9      w(w<=0.7) = 0;
10     w=wOrg + w;
11 end
12 w=~w;
13 w = double(w);
14 w = bwmorph(w, 'thin',100);
15 w = double(w);
16 skeletonizedImage = ~w;
17 w1 = w(2:end-1,2:end-1);
18 w2 = w(1:end-2,2:end-1);
19 w3 = w(1:end-2,3:end);
20 w4 = w(2:end-1,3:end);
21 w5 = w(3:end,3:end);
22 w6 = w(3:end,2:end-1);
23 w7 = w(3:end,1:end-2);
24 w8 = w(2:end-1,1:end-2);
25 w9 = w(1:end-2,1:end-2);
26 id1 = w1+w2+w4+w6+w8==4 & w1==1;
27 id2 = w1+w2+w4+w6+w8==5  & w1==1;
28 id3 =(w1+w2+w4+w7)==4  & w1==1;
29 id4 = (w1+w4+w6+w9)==4  & w1==1;
30 id5 = (w1+w6+w8+w3)==4  & w1==1;
31 id6 = (w1+w2+w8+w5)==4  & w1==1;
32 id7 = (w1+w3+w5+w8-w4)==4  & w1==1;
33 id8 = (w1+w5+w7+w2-w6)==4  & w1==1;
34 id9 = (w1+w7+w9+w4-w8)==4  & w1==1;
35 id10 = (w1+w9+w3+w6-w2)==4  & w1==1;
36 id11 = (w1+w3+w5+w7+w9)==4  & w1==1;
37 id12 = (w1+w3+w5+w7+w9)==5  & w1==1;
38 id13 = w2+w3+w4+w5+w6+w7+w8+w9==1 & w1==1;
39 id = id1+id2+id3+id4+id5+id6+id7+id8+id9+id10+id11+id12+id13;
40 [r,c] = find(id>=1);
41 r=r+1;
42 c=c+1;
```

```
43  finalJoints=[];
44  while ~isempty(r)
45      rCP = r(1);
46      cCP = c(1);
47      d = sqrt((r-rCP).^2+(c-cCP).^2);
48      id = d<=3;
49      finalJoints = [finalJoints; mean(r(id)) mean(c(id))];    %#ok
50      r(id)=[];
51      c(id)=[];
52  end
53  finalJoints = round(unique(finalJoints,'rows'));
54  junctions.c =  finalJoints(:,2);
55  junctions.r =  finalJoints(:,1);
56  t = linspace(0,2*pi,20);
57  x = 1.5*cos(t);
58  y = 1.5*sin(t);
59  junctions.surround.x = zeros(length(junctions.c),20);
60  junctions.surround.y = zeros(length(junctions.r),20);
61  for i = 1 : length(junctions.c)
62      junctions.surround.x(i,:)= x+ junctions.c(i);
63      junctions.surround.y(i,:) = y+ junctions.r(i);
64  end
65  end
```

---

## Code S6: Parallel Vein Detection (PVD)

This function detects the venation pattern inside the wing using the parallel method introduced in the manuscript.
**Inputs:** This function has two inputs. Both inputs are generated by the *junctionDetection* function (Code S5). These inputs are as follows:

- *skeletonizedImage*
- *junctions*

**Outputs:** This function has one output as follows:

- *detectedVeins*: This variable is a structure with the following fields:
  - *detectedVeins.coordinate*: A 1-by-N cell array. N represents the number of detected veins. Each cell of this array contains the coordinate of the vein path.
  - *detectedVeins.length*: A 1-by-N array storing the length of veins. This variable is used in plotting a histogram.
  - *detectedVeins.Color*: An N-by-3 matrix that stores the corresponding color of each vein considering the vein length. These colors are used to plot a heatmap of the venation pattern.

**Code Description:**

- **Lines 7-16:** This code is a modified path-finding method that is adopted to detect the path between junctions (venation pattern). The initial point for starting to find a path is the junctions that are detected in Code S5. Lines 7 and 8 show these initial points. Each initial point detects new points on the path. In this code, only venations are white (pixel value = 1), and the rest are black (pixel value = 0). This code defines a variable named *front*. This variable stores the data related to the new points detected on the venation path. This variable is a structure with the following fields:

  - *front.Index*: Stores the index of front points. We used this variable to change the color of front points easily (Line 10). This avoids the code from checking already detected points on the path.
  - *front.c*: Stores the column index (x coordinate) of front points.
  - *front.r*: Stores the row index (y coordinate) of front points.
  - *front.parent*: Each new detected point on the path of the vein should know its origin (parent). In this case, when the path starts to be detected from one joint and it reaches another junction, which means that the path is already detected, the very last front point that meets a new junction finds its way back from its parents until it reaches the initial point. This means the path between those junctions is extracted. This variable has also two fields as follows.
    * *front.parent.c*: The column index (x coordinate) of the parent of the front points.
    * *front.parent.r*: The row index (y coordinate) of the parent of the front points.

- **Lines 17-93:** Line 17 starts a while loop in which the initial point from all junctions parallelly starts finding their path on the skeletonized image of the wing. In lines 18-35, the front points are defined. In lines 36-39, their parents are defined. In lines 41-48, all repetitive front points are removed. In lines 49-70, all front points that are not on the path (out of the path in the black area or in the area of the detected path, which is gray) are removed. In lines 70-92, the code checks if fronts from different paths meet each other or not. This while loop continues until no white pixels are inside the wing. *WingSegment Video S4* in the supplementary material visualizes this process.

- **Lines 117-156:** After all fronts meet each other (technically in the middle of each vein), another while loop extracts the venation pattern. In this case, each vein is divided into two halves. The direction of points on these two halves is opposite. Then the first half extracts in lines 108-125. Then the second half extracts in lines 126-143. Then, in lines 144-55, the second half reverses and merges with the first half, and also the length of veins is measured in lines 149-154.

- **Lines 157-162:** In these lines, the code finds the color of each vein corresponding to its length from the jet colormap.

Code S6: Parllel Vein Detection code in MATLAB, for automated vein detecting using parallel detection method

```
1  function detectedVeins = veinDetection_PVD(skeletonizedImage,junctions)
2  w = double(~skeletonizedImage);
3  joints = junctions;
4  image.width = size(w,2);
5  image.height = size(w,1);
6  %%
7  initialPoint.c =  joints.c';
8  initialPoint.r =  joints.r';
9  front.Index = ((initialPoint.c-1)*image.height)+initialPoint.r;
10 w(front.Index) = 0.3;
11 detectedPoints.c = initialPoint.c;
12 detectedPoints.r = initialPoint.r;
13 detectedPoints.parent.c = zeros(1,length(initialPoint.c));
14 detectedPoints.parent.r = zeros(1,length(initialPoint.c));
15 meetingPoints.main = [];
16 meetingPoints.parent = [];
```

```
while any(any(w==1)) && ~isempty( initialPoint.c)
    front.c = [...
        initialPoint.c+1,...        %Column Index: Right Pixel
        initialPoint.c+1,...        %Column Index: UpRight Pixel
        initialPoint.c,...            %Column Index: Up Pixel
        initialPoint.c-1,...        %Column Index: UpLeft Pixel
        initialPoint.c-1,...        %Column Index: Left Pixel
        initialPoint.c-1,...        %Column Index: LeftBottom Pixel
        initialPoint.c,...            %Column Index: Bottom Pixel
        initialPoint.c+1];          %Column Index: BottomRight Pixel
    front.r = [...
        initialPoint.r,...            %Column Index: Right Pixel
        initialPoint.r-1,...        %Column Index: UpRight Pixel
        initialPoint.r-1,...        %Column Index: Up Pixel
        initialPoint.r-1,...        %Column Index: UpLeft Pixel
        initialPoint.r,...            %Column Index: Left Pixel
        initialPoint.r+1,...        %Column Index: LeftBottom Pixel
        initialPoint.r+1,...        %Column Index: Bottom Pixel
        initialPoint.r+1];          %Column Index: BottomRight Pixel
    front.parent.c = repmat(initialPoint.c,1,8);
    front.parent.c = front.parent.c(:)';
    front.parent.r =  repmat(initialPoint.r,1,8);
    front.parent.r = front.parent.r(:)';
    front.Index = ((front.c-1)*image.height)+front.r;
    %% Unique front
    Front = [front.c' front.r' front.parent.c' front.parent.r'];
    [~,iUnique,~] = unique(Front,'rows');
    front.c = front.c(iUnique);
    front.r = front.r(iUnique);
    front.parent.c = front.parent.c(iUnique);
    front.parent.r = front.parent.r(iUnique);
    front.Index = front.Index(iUnique);
    %% Remove out of bound nodes of the front
    id1_outOfBount = front.Index < 1;
    id2_outOfBount = front.Index > numel(w);
    front.c (id1_outOfBount) = [];
    front.r (id1_outOfBount) = [];
    front.parent.c(id1_outOfBount) = [];
    front.parent.r(id1_outOfBount) = [];
    front.Index (id1_outOfBount) = [];
    front.c (id2_outOfBount) = [];
    front.r (id2_outOfBount) = [];
    front.parent.c(id2_outOfBount) = [];
    front.parent.r(id2_outOfBount) = [];
    front.Index (id2_outOfBount) = [];
    %% Remove black/gray pixels of the front
    id_black =w(front.Index) ~= 1;
    front.c (id_black) = [];
    front.r (id_black) = [];
    front.parent.c(id_black) = [];
    front.parent.r(id_black) = [];
    front.Index (id_black) = [];
    w(front.Index) = 0.3;
    %%
    detectedPoints.c = [detectedPoints.c     front.c];
    detectedPoints.r = [detectedPoints.r      front.r];
    detectedPoints.parent.c = [detectedPoints.parent.c    front.parent.c];
    detectedPoints.parent.r = [detectedPoints.parent.r     front.parent.r];
    %%
    [x,y] = meshgrid(1:length(front.c));
    X = x(x~=y);
    Y = y(x~=y);
    d1= sqrt(((front.c(X(:))-front.c(Y(:))).^2) + ((front.r(X(:))-front.r(Y(:))).^2))';
    d2= sqrt(((front.parent.c(X(:))-front.parent.c(Y(:))).^2) + ((front.parent.r(X(:))-
        front.parent.r(Y(:))).^2))';
    id = find (d1 <1.5 & d2>1.5);
    if ~isempty(id)
        for i = 1:length(id)
            meetingPoints.main(end+1,1:4) = [front.c(X(id(i)))       front.r(X(id(i)))
                    front.c(Y(id(i)))       front.r(Y(id(i)))];
            meetingPoints.parent(end+1,1:4) = [front.parent.c(X(id(i)))       front.
                parent.r(X(id(i)))       front.parent.c(Y(id(i)))       front.parent.r(
                Y(id(i)))];
```

```
86          end
87          id_Siblings =  find((meetingPoints.parent(:,1) == meetingPoints.parent(:,3)) & (
                meetingPoints.parent(:,2) == meetingPoints.parent(:,4)));
88          meetingPoints.parent(id_Siblings,:) = [];
89          meetingPoints.main(id_Siblings,:) = [];
90      end
91      initialPoint.c = front.c;
92      initialPoint.r = front.r;
93  end
94  id_Siblings =  (meetingPoints.parent(:,1) == meetingPoints.parent(:,3)) & (meetingPoints
        .parent(:,2) == meetingPoints.parent(:,4));
95  meetingPoints.parent(id_Siblings,:) = [];
96  meetingPoints.main(id_Siblings,:) = [];
97  [meetingPoints.main,id,~] = unique(meetingPoints.main,'rows');
98  meetingPoints.parent = meetingPoints.parent(id,:);
99  s1 = [mean(meetingPoints.main(:,[1  3]),2)     mean(meetingPoints.main(:,[2  4]),2)];
100 s2 = [mean(meetingPoints.parent(:,[1  3]),2)     mean(meetingPoints.parent(:,[2  4]),2)];
101 d = sqrt(sum((s1'-s2').^2));
102 meetingPoints.main = meetingPoints.main(d<=0.5,:);
103 meetingPoints.parent = meetingPoints.parent(d<=0.5,:);
104 MeetingPoints_backUp = meetingPoints.main;
105 detectedVeins.coordinate = {};
106 detectedVeins.length = [];
107 while ~isempty(MeetingPoints_backUp)
108     %% FirstHalf
109     id = find(detectedPoints.c==MeetingPoints_backUp(1,1) & detectedPoints.r==
            MeetingPoints_backUp(1,2),1);
110     path1.c = [detectedPoints.c(id) detectedPoints.parent.c(id)];
111     path1.r = [detectedPoints.r(id)  detectedPoints.parent.r(id)];
112     detectedPoints.c(id) = [];
113     detectedPoints.r(id)  = [];
114     detectedPoints.parent.c(id)  = [];
115     detectedPoints.parent.r(id)  = [];
116     isDetected = false;
117     while ~isDetected && ~isempty(path1.c) && ~isempty(id)
118         id = find(detectedPoints.c==path1.c(end) & detectedPoints.r==path1.r(end),1);
119         if detectedPoints.parent.c(id)==0
120             isDetected = true;
121         elseif ~isempty(id)
122             path1.c(end+1) = detectedPoints.parent.c(id);
123             path1.r(end+1) = detectedPoints.parent.r(id);
124         end
125     end
126     %% SecondHalf
127     id = find(detectedPoints.c==MeetingPoints_backUp(1,3) & detectedPoints.r==
            MeetingPoints_backUp(1,4),1);
128     path2.c = [detectedPoints.c(id) detectedPoints.parent.c(id)];
129     path2.r = [detectedPoints.r(id)  detectedPoints.parent.r(id)];
130     detectedPoints.c(id) = [];
131     detectedPoints.r(id)  = [];
132     detectedPoints.parent.c(id)  = [];
133     detectedPoints.parent.r(id)  = [];
134     isDetected = false;
135     while ~isDetected && ~isempty(path2.c) && ~isempty(id)
136         id = find(detectedPoints.c==path2.c(end) & detectedPoints.r==path2.r(end),1);
137         if detectedPoints.parent.c(id)==0
138             isDetected = true;
139         elseif ~isempty(id)
140             path2.c(end+1) = detectedPoints.parent.c(id);
141             path2.r(end+1) = detectedPoints.parent.r(id);
142         end
143     end
144     %% MergeHalf
145     detectedPath = [...
146         path1.c(end:-1:1) path2.c;...
147         path1.r(end:-1:1) path2.r];
148     detectedPath = unique(detectedPath','rows','stable')';
149     if ~isempty( detectedPath)
150         [latout,lonout] = reducem(detectedPath(1,:)',detectedPath(2,:)',1.5);
151         detectedVeins.coordinate(end+1) = {[latout lonout]};
152         d = sqrt(((latout(2:end)-latout(1:end-1)).^2)+((lonout(2:end)-lonout(1:end-1))
                .^2));
153         detectedVeins.length(end+1) = sum(d);
```

```
154         end
155         MeetingPoints_backUp(1,:) = [];
156     end
157     Data = detectedVeins.length;
158     maxData = max(max(detectedVeins.length));
159     minData = min(min(detectedVeins.length));
160     colorMap = jet;
161     idColor = round(1+(((Data-minData)/(maxData-minData))*255));
162     detectedVeins.Color = colorMap(idColor,:);
163     end
```

## Code S7: Sequential Vein Detection (SVD)

Code S7 is the code that detects junctions using the sequential method. The inputs and outputs of this function are similar to those in Code S6.
**Code Description:** Some parts of this code are different from PVD; as a result, we only describe where necessary.

- **Line 9:** This line starts a *while* loop, which refers to the junctions. This loop lets the code continue its work until all junctions are involved in vein detection. In this method, in contrast to the PVD method, in each step, only one junction searches all paths that are connected to that junction until it reaches other junctions. This loop continues until all junctions are involved.
- **Line 26:** This line starts another *while* loop which is located inside the previous loop. This loop relates to the individual junction, which is searching connected paths. This while loop continues until all paths connected to that junction are checked.
- **Lines 71-100:** These lines in each step check whether the front points meet another junction. It should be mentioned that normally, several veins are connected to each junction. This means each junction is checking several paths at the same time to find other new junctions. *WingSegment Video S5* in the supplementary information visualizes this process.
- **Lines 132-156:** When one individual junction has detected all connected junctions, this *for* loop finds the path between those junctions by tracking their parents.

Code S7: Sequential Vein Detection (SVD) code in MATLAB, for automated vein detecting using parallel detection method

```
1   function detectedVeins = veinDetection_SVD(skeletonizedImage,junctions)
2   joints = junctions;
3   w = double(~skeletonizedImage);
4   image.width = size(w,2);
5   image.height = size(w,1);
6   detectedVeins = {};
7   detectedVeins.coordinate = {};
8   detectedVeins.length = [];
9   while ~isempty(joints.c)
10      initialPoint.c =  joints.c(1);
11      initialPoint.r =  joints.r(1);
12      endingPoint.c = [];
13      endingPoint.r = [];
14      endingPoint.parent.r = [];
15      endingPoint.parent.c = [];
16      w(initialPoint.r,initialPoint.c) = 0.3;
17      joints.c(1,:) = [];
18      joints.r(1,:) = [];
```

```
19      joints.surround.x(1,:) = [];
20      joints.surround.y(1,:) = [];
21      detectedPoints.c = initialPoint.c;
22      detectedPoints.r = initialPoint.r;
23      detectedPoints.parent.c = 0;
24      detectedPoints.parent.r = 0;
25      isDetected = false;
26      while ~isDetected
27          %% Define front pixels
28          front.c = [...
29              initialPoint.c+1,...        %Column Index: Right Pixel
30              initialPoint.c+1,...        %Column Index: UpRight Pixel
31              initialPoint.c,...            %Column Index: Up Pixel
32              initialPoint.c-1,...          %Column Index: UpLeft Pixel
33              initialPoint.c-1,...          %Column Index: Left Pixel
34              initialPoint.c-1,...          %Column Index: LeftBottom Pixel
35              initialPoint.c,...             %Column Index: Bottom Pixel
36              initialPoint.c+1];           %Column Index: BottomRight Pixel
37          front.r = [...
38              initialPoint.r,...              %Column Index: Right Pixel
39              initialPoint.r-1,...          %Column Index: UpRight Pixel
40              initialPoint.r-1,...          %Column Index: Up Pixel
41              initialPoint.r-1,...          %Column Index: UpLeft Pixel
42              initialPoint.r,...             %Column Index: Left Pixel
43              initialPoint.r+1,...        %Column Index: LeftBottom Pixel
44              initialPoint.r+1,...        %Column Index: Bottom Pixel
45              initialPoint.r+1];           %Column Index: BottomRight Pixel
46          front.parent.c = repmat(initialPoint.c,1,8);
47          front.parent.c = front.parent.c(:)';
48          front.parent.r =  repmat(initialPoint.r,1,8);
49          front.parent.r = front.parent.r(:)';
50          front.Index = ((front.c-1)*image.height)+front.r;
51        %% Remove out of bound nodes of the front
52          id1_outOfBount = front.Index < 1;
53          id2_outOfBount = front.Index > numel(w);
54          front.c (id1_outOfBount) = [];
55          front.r (id1_outOfBount) = [];
56          front.parent.c(id1_outOfBount) = [];
57          front.parent.r(id1_outOfBount) = [];
58          front.Index (id1_outOfBount) = [];
59          front.c (id2_outOfBount) = [];
60          front.r (id2_outOfBount) = [];
61          front.parent.c(id2_outOfBount) = [];
62          front.parent.r(id2_outOfBount) = [];
63          front.Index (id2_outOfBount) = [];
64        %% Remove gray pixels of the front
65          id_notWhite =w(front.Index) ~= 1;
66          front.c (id_notWhite) = [];
67          front.r (id_notWhite) = [];
68          front.parent.c(id_notWhite) = [];
69          front.parent.r(id_notWhite) = [];
70          front.Index (id_notWhite) = [];
71         %% Check if front is in junction area
72          for iJunctions = 1 : length(joints.c)
73              xv = joints.surround.x(iJunctions,:);
74              yv = joints.surround.y(iJunctions,:);
75              xq.c = front.c;
76              yq.r = front.r;
77              xq.parent = front.parent.c;
78              yq.parent = front.parent.r;
79              removeFront = [];
80              for iFront = 1 : length(xq.c)
81                  in = inpolygon(xq.c(iFront),yq.r(iFront),xv,yv);
82                  % Save detected points if are inside junction area
83                  if any(in)
84                      detectedPoints.c =[detectedPoints.c    xq.c(iFront)     joints.c(
                            iJunctions)];
85                      detectedPoints.r = [detectedPoints.r    yq.r(iFront)     joints.r(
                            iJunctions)]; %*ones(1,sum(in))
86                      detectedPoints.parent.c = [detectedPoints.parent.c  xq.parent(iFront
                            )     xq.c(iFront) ];
87                      detectedPoints.parent.r = [detectedPoints.parent.r   yq.parent(
                            iFront)     yq.r(iFront) ];
```

```
88                      removeFront = [removeFront iFront]; %#ok
89                      endingPoint.c = [endingPoint.c       joints.c(iJunctions)];
90                      endingPoint.r = [endingPoint.r        joints.r(iJunctions)];
91                      endingPoint.parent.c = [endingPoint.parent.c     xq.parent(iFront)];
92                      endingPoint.parent.r = [endingPoint.parent.r      yq.parent(iFront)];
93                  end
94              end
95              front.c(removeFront) = [];
96              front.r(removeFront) = [];
97              front.parent.c(removeFront) = [];
98              front.parent.r(removeFront) = [];
99              front.Index(removeFront) = [];
100         end
101         %% Save detected points
102         detectedPoints.c =[detectedPoints.c front.c ];
103         detectedPoints.r = [detectedPoints.r front.r ];
104         detectedPoints.parent.c = [detectedPoints.parent.c  front.parent.c];
105         detectedPoints.parent.r = [detectedPoints.parent.r   front.parent.r];
106         if isempty(front.c)
107             isDetected = true;
108         end
109         w(front.Index) = 0.3;
110         initialPoint.c = front.c;
111         initialPoint.r = front.r;
112     end
113     if ~isempty(endingPoint.c)
114         allEnding = [
115             endingPoint.c
116             endingPoint.r
117             endingPoint.parent.c
118             endingPoint.parent.r];
119         ckeckEnding = unique(allEnding','rows');
120         endingPoint.c = ckeckEnding(:,1)';
121         endingPoint.r = ckeckEnding(:,2)';
122         allDetectedPoints = [
123             detectedPoints.c
124             detectedPoints.r
125             detectedPoints.parent.c
126             detectedPoints.parent.r];
127         ckeckDetected = unique(allDetectedPoints','rows');
128         detectedPoints.c = ckeckDetected(:,1)';
129         detectedPoints.r = ckeckDetected(:,2)';
130         detectedPoints.parent.c = ckeckDetected(:,3)';
131         detectedPoints.parent.r = ckeckDetected(:,4)';
132         for i= 1: length(endingPoint.c)
133             id = find(detectedPoints.c==endingPoint.c(i) & detectedPoints.r ==
                    endingPoint.r(i),1);
134             path.c = [detectedPoints.c(id) detectedPoints.parent.c(id)];
135             path.r = [detectedPoints.r(id)  detectedPoints.parent.r(id)];
136             detectedPoints.c(id) = [];
137             detectedPoints.r(id)  = [];
138             detectedPoints.parent.c(id)  = [];
139             detectedPoints.parent.r(id)  = [];
140             isDetected = false;
141             while ~isDetected && ~isempty(path.c)
142                 id = find(detectedPoints.c==path.c(end) & detectedPoints.r == path.r(end
                        ),1);
143                 if detectedPoints.parent.c(id)==0
144                     isDetected = true;
145                 else
146                     path.c(end+1) = detectedPoints.parent.c(id);
147                     path.r(end+1) = detectedPoints.parent.r(id);
148                 end
149             end
150             if ~isempty(path.c)
151                 [latout,lonout] = reducem(path.c',path.r',1);
152                 detectedVeins.coordinate(end+1) = {[latout lonout]};
153                 d = sqrt(((latout(2:end)-latout(1:end-1)).^2)+((lonout(2:end)-lonout(1:
                        end-1)).^2));
154                 detectedVeins.length(end+1) = sum(d);
155             end
156         end
157     end
```

```
end
Data = detectedVeins.length;
maxData = max(max(detectedVeins.length));
minData = min(min(detectedVeins.length));
colorMap = jet;
idColor = round(1+(((Data-minData)/(maxData-minData))*255));
detectedVeins.Color = colorMap(idColor,:);
end
```

---

## Code S8: FreeCAD Macro File Generation

This function is developed to generate a FreeCAD macro file.
**Inputs:** This function has two inputs as follows:

- *wingOutline*: This variable is exactly the 4th output of Code S4.
- *wingCells*: This variable is exactly the 5th output of Code S4.

**Output:** The only output of this function is an FCMacro file.
**Code Description:**

- **Lines 22-24:** This loop defines the wing outline.
- **Lines 31-42:** These loops define the wing cell domains.

Code S8: FreeCAD Macro File Generation code in MATLAB

```
function generateFreeCadModel(wingOutline,wingCells)
[file,path] = uiputfile('*.FCMacro');
fid = fopen([path file],'w');
fprintf(fid,'import FreeCAD\n');
fprintf(fid,'import Draft\n');
fprintf(fid,'App.newDocument("Wing")\n');
fprintf(fid,'def makewire(Name,PointList,Placement= False,Closed=False,Support=None,Face=None):\n');
fprintf(fid,'    WireObj = FreeCAD.ActiveDocument.addObject("Part::Part2DObjectPython",Name)\n');
fprintf(fid,'    Draft._Wire(WireObj)\n');
fprintf(fid,'    WireObj.Points = PointList\n');
fprintf(fid,'    WireObj.Closed = Closed\n');
fprintf(fid,'    WireObj.Support = Support\n');
fprintf(fid,'    if Face != None:\n');
fprintf(fid,'        WireObj.MakeFace = Face\n');
fprintf(fid,'    if Placement: WireObj.Placement = Placement\n');
fprintf(fid,'    if Gui:\n');
fprintf(fid,'        Draft._ViewProviderWire(WireObj.ViewObject)\n');
fprintf(fid,'        Draft.formatObject(WireObj)\n');
fprintf(fid,'        Draft.select(WireObj)\n');
fprintf(fid,'    FreeCAD.ActiveDocument.recompute()\n');
fprintf(fid,'points=[\n');
for i = 1: length(wingOutline)
    fprintf(fid,'FreeCAD.Vector(%g,%g,0.0),\n',wingOutline(i,1),wingOutline(i,2));
end
fprintf(fid,'           ]\n');
fprintf(fid,'Pl = App.Placement(Pos =(0,0,0))\n');
fprintf(fid,'Pl.Rotation.Q = (0.0, 0.0, 0.0, 1.0)\n');
fprintf(fid,'Pl.Base = points[0]\n');
fprintf(fid,'makewire("Outline", points,Placement= Pl)\n');
fprintf(fid,'FreeCAD.getDocument(''Wing'').getObject(''Outline'').Placement = App.Placement(App.Vector(0.00,0.00,0.00),App.Rotation(App.Vector(0.00,0.00,1.00),0.00))\n');
for j = 1 : length(wingCells)
    fprintf(fid,'points=[\n');
    for i = 1: length(wingCells{j})
        fprintf(fid,'FreeCAD.Vector(%g,%g,0.0),\n',wingCells{j}(i,1),wingCells{j}(i,2));
```

```
35      end
36      fprintf(fid,'            ]\n');
37      fprintf(fid,'Pl = App.Placement(Pos =(0,0,0))\n');
38      fprintf(fid,'Pl.Rotation.Q = (0.0, 0.0, 0.0, 1.0)\n');
39      fprintf(fid,'Pl.Base = points[0]\n');
40      fprintf(fid,'makewire("%s", points,Placement= Pl)\n',['Cell' num2str(j)]);
41      fprintf(fid,'FreeCAD.getDocument(''Wing'').getObject(''%s'').Placement = App.
            Placement(App.Vector(0.00,0.00,0.00),App.Rotation(App.Vector(0.00,0.00,1.00)
            ,0.00))\n',['Cell' num2str(j)]);
42  end
43  fclose(fid);
44  end
```

## Code S9: Run Region Growing

We developed this code to demonstrate how to use the *RegionGrowing* function in projects. In this code, users can mark a closed area inside the wing when importing an image. The code prints the coordinates of the boundary of that extracted cell.
**Input:** The input for this function is a binary image of the wing, which is obtained using Code S2 in line 3 for this purpose.
**Output:** The output of this function is *boundary*, which is an N-by-2 array containing the coordinates of the boundary of the selected region.
**Code Description:**

- **Line 3:** This line runs the *GetImage* function described in Code S2.
- **Lines 5-11:** In these lines, the code displays the imported image and waits for the user to click inside one region.
- **Line 12:** This line runs the *RegionGrowing* function, using the point chosen by the user by clicking inside the region as the initial point.
- **Lines 13-17:** These lines visualize the detected boundary and display the image after being processed by the *RegionGrowing* function.

Code S9: Developed code for running Region Growing code

```
1   function boundary = runRegionGrowing
2   %% Import Image / Binarization
3   [imageBinaryImage,~,~,imageHeight]=GetImage;
4   %%  Region Growing
5   fig = figure('Name','Region Growing');
6   ax =gca(fig);
7   imshow(imageBinaryImage,'parent',ax)
8   title(ax,'Please click inside a desired domain')
9   roi = drawpoint(ax);
10  c = round(roi.Position(1));
11  r = round(roi.Position(2));
12  [~,n_white,im,boundary]=RegionGrowing(imageBinaryImage,r,c);
13  cla(ax)
14  imshow(im,'parent',ax)
15  hold(ax,'on')
16  plot(boundary(:,1),imageHeight-boundary(:,2),'color','r','linewidth',3,'linestyle','-','
        parent',ax)
17  title(ax,['Area: ' num2str(n_white) 'px^2 '])
18  end
```

## Code S10: Run WingSegment

This code is developed to demonstrate how WingSegment integrates various components to segment an insect wing image.
**Code Description:**

- **Line 3:** To initiate wing image segmentation, the first function is *GetImage* (Code S2).
- **Line 5:** If we begin with extracting junctions, the next code is *junctionDetection* (Code S5). This function uses *imageBinaryImage* as input from the *GetImage* function.
- **Lines 7-14:** These lines create the first figure and plot detected junctions.
- **Lines 16-18:** These lines visualize the skeletonized image.
- **Line 20:** In this line, the *veinDetection_SVD* function runs. It should be noted that prior to vein detection, junction detection should be executed, as vein detection requires the location of junctions and the skeletonized image as input.
- **Lines 22-31:** These lines visualize detected veins using the SVD method.
- **Lines 33-42:** These lines also plot a heatmap of vein lengths detected with the SVD method.
- **Lines 44-46:** These lines generate a histogram related to the vein lengths extracted from the SVD method.
- **Lines 48-74:** These lines are analogous to lines 20-46 but for the PVD method.
- **Lines 76-81:** These lines illustrate the *cellSegment* function. For this function, the *GetImage* function needs to be executed beforehand because it requires the binary image as input.
- **Lines 83-96:** In these lines, we demonstrate how to generate heatmaps of the wing cells' area, length, and circularity distribution.
- **Lines 98-108:** These lines show how one can plot the extracted wing cells.
- **Line 110:** This line runs the *generateFreeCadModel* function. Note that this function can only be executed after extracting cells, as it requires *wingOutline* and *wingCells* as input.

Code S10: Integerating all functions to segment the wing

```
1  function runWingSegment
2  %% Import Image / Binarization
3  [imageBinaryImage,~,~,~]=GetImage;
4  %% Junction Detection / Wing Skeletonization
5  [junctions, skeletonizedImage]=junctionDetection(imageBinaryImage);
6  % Figure 1: Detected Junctions
7  fig1 = figure('Name','Detected Junctions');
8  ax1 = gca(fig1);
9  imshow(imageBinaryImage,'parent',ax1)
10 hold(ax1,'on')
11 xJunctions = junctions.c;
12 yJunctions = junctions.r;
13 plot(xJunctions, yJunctions, 'marker','o','markeredgecolor','r','markerfacecolor','r','
       markersize',10,'parent',ax1,'linestyle','none')
```

```
axis(ax1,'equal')
% Figure 2: 'Skeletonized Image'
fig2 = figure('Name','Skeletonized Image');
ax2 = gca(fig2);
imshow(skeletonizedImage,'parent',ax2)
%% Sequential Vein Detection (SVD)
detectedVeins_SVD = veinDetection_SVD(skeletonizedImage,junctions);
% Figure 3: 'Detected Veins (SVD)'
fig3 = figure('Name','Detected Veins (SVD)');
ax3 =gca(fig3);
nVeins = length(detectedVeins_SVD.coordinate);
hold(ax3,'on')
for i = 1 : nVeins
    xVeins = detectedVeins_SVD.coordinate{i}(:,1);
    yVeins = size(imageBinaryImage,1)-detectedVeins_SVD.coordinate{i}(:,2);
    plot(xVeins,yVeins,'linestyle','-','linewidth',3,'color','b','parent',ax3)
end
axis(ax3,'equal')
% Figure 4: 'Detected Veins (SVD)-Length Distribution)'
fig4 = figure('Name','Detected Veins (SVD)-Length Distribution');
ax4 =gca(fig4);
hold(ax4,'on')
for i = 1 : nVeins
    veinColor = detectedVeins_SVD.Color(i,:);
    xVeins = detectedVeins_SVD.coordinate{i}(:,1);
    yVeins = size(imageBinaryImage,1)-detectedVeins_SVD.coordinate{i}(:,2);
    plot(xVeins,yVeins,'linestyle','-','linewidth',3,'color',veinColor,'parent',ax4)
end
axis(ax4,'equal')
% Figure 5: Detected Veins (SVD)-Length Distribution Histogram
fig5 = figure('Name','Detected Veins (SVD)-Length Distribution Histogram');
ax5 =gca(fig5);
histogram(ax5,detectedVeins_SVD.length,10)
%% Parallel Vein Detection (PVD)
detectedVeins_PVD = veinDetection_PVD(skeletonizedImage,junctions);
% Figure 6: Detected Veins (PVD)
fig6 = figure('Name','Detected Veins (PVD)');
ax6 =gca(fig6);
nVeins = length(detectedVeins_PVD.coordinate);
hold(ax6,'on')
for i = 1 : nVeins
    xVeins = detectedVeins_PVD.coordinate{i}(:,1);
    yVeins = size(imageBinaryImage,1)-detectedVeins_PVD.coordinate{i}(:,2);
    plot(xVeins,yVeins,'linestyle','-','linewidth',3,'color','b','parent',ax6)
end
axis(ax6,'equal')
% Figure 7: Detected Veins (PVD)-Length Distribution
fig7 = figure('Name','Detected Veins (PVD)-Length Distribution');
ax7 =gca(fig7);
hold(ax7,'on')
for i = 1 : nVeins
    veinColor = detectedVeins_PVD.Color(i,:);
    xVeins = detectedVeins_PVD.coordinate{i}(:,1);
    yVeins = size(imageBinaryImage,1)-detectedVeins_PVD.coordinate{i}(:,2);
    plot(xVeins,yVeins,'linestyle','-','linewidth',3,'color',veinColor,'parent',ax7)
end
axis(ax7,'equal')
% Figure 8: Detected Veins (PVD)-Length Distribution Histogram
fig8 = figure('Name','Detected Veins (PVD)-Length Distribution Histogram');
ax8 =gca(fig8);
histogram(ax8,detectedVeins_PVD.length,10)
%% Cell Segmentation
[ContourMatrix_Area,...
    ContourMatrix_Length,...
    ContourMatrix_Circularity,...
    wingOutline,...
    wingCells,...
    wingInfo] = cellSegment(imageBinaryImage);
% Figure 9: Area Distribution Filled Contour
fig9 = figure('Name','Wing Cells Area Distribution Filled Contour');
ax9 =gca(fig9);
contourf(ax9,ContourMatrix_Area,100,'linestyle','none')
axis(ax9,'equal')
```

```
87  % Figure 10: Wing Cells Length Distribution Filled Contour
88  fig10 = figure('Name','Wing Cells Length Distribution Filled Contour');
89  ax10 =gca(fig10);
90  contourf(ax10,ContourMatrix_Length,100,'linestyle','none')
91  axis(ax10,'equal')
92  % Figure 11: Wing Cells Circularity Distribution Filled Contour
93  fig11 = figure('Name','Wing Cells Circularity Distribution Filled Contour');
94  ax11 =gca(fig11);
95  contourf(ax11,ContourMatrix_Circularity,100,'linestyle','none')
96  axis(ax11,'equal')
97  % Figure 12: Extracted Wing Cells
98  fig12 = figure('Name','Extracted Wing Cells');
99  ax12 =gca(fig12);
100 plot(wingOutline(:,1),wingOutline(:,2),'linewidth',2,'linestyle','-','color','b','parent
        ',ax12)
101 hold(ax12,'on')
102 nCells = length(wingCells);
103 for i = 1 : nCells
104     x = wingCells{i}(:,1);
105     y = wingCells{i}(:,2);
106     plot(x,y,'linewidth',2,'color','r','linestyle','-','parent',ax12,'linewidth',2)
107 end
108 axis(ax12,'equal')
109 %% Generate FreeCad Model
110 generateFreeCadModel(wingOutline,wingCells)
111 end
```

---

## Code S11: Automated Superimposition Using Particle Swarm Optimization (PSO)

This code is developed for automated superimposing wings. The code consists of four functions.

1. ***PSO_Superimposition.*** This is the main function of the PSO algorithm specifically developed for automated superimposing.

   **Inputs.** This function has two inputs.

   - *outline_im1*. This variable refers to the extracted outline from the first image.
   - *outline_im2*. This variable refers to the extracted outline from the second image.

   **Output**. *BestParticle* is the only output of this function. It is a 1-by-3 array in which the first and second elements refer to the x and y coordinates of the translation of the first image, and the third element relates to the rotation of the second.

   **Code Description.**

   - **Lines 2-6.** The number of variables for each particle in the PSO algorithm and their boundary conditions are defined in these lines. Each particle consists of three variables. The first and second refer to the x and y coordinates for translating the wing. The third one refers to the angle of wing rotation.
   - **Lines7-35.** PSO parameters are defined in these lines. The number of particles is defined in Line 9, and the maximum number of iterations is defined in Line 8. Besides, the initial particles are defined in the for loop in Lines 23-35.
   - **Lines 36-57:** These lines are the main loop of the PSO.

2. ***costFun.*** This function is called in lines 27 and 46 of the *PSO_Superimposition* function. It calculates the distance between wing outlines following the repositioning of the second image, considering the translation and rotation of an individual particle. A penalty function in Lines 74-76 is the penalty for an individual that induces an angle of more than 10 degrees between two wings.

3. ***distanceFun.*** This function is called in Line 69 of the *costFun* to calculate the distance between the first wing and the second wing after translation and rotation.

4. ***wingsAngle.*** This function is called in Lines 72 and 73 to measure the angle of each wing concerning the horizontal axis line.

Code S11: Particle Swarm Optimization Code Developed for Wings Automated Superimposition

```
1  function BestParticle=PSO_Superimposition(outline_im1,outline_im2)
2  %% Problem Definition
3  nVar = 3;
4  varSize = [1 nVar];
5  varMin  = [-10 -10  -90];
6  varMax =  [ 10  10   90];
7  %% PSO Parameters
8  maxIt = 40;
9  nParticle = 30;
10 w = 1;
11 wDamp = 0.95;
12 c1 = 1.2;
13 c2 = 1.2;
14 %% Initialization
15 Particle.Position = zeros(varSize);
16 Particle.Cost = [];
17 Particle.Velocity = [];
18 Particle.Best.Position =  [];
19 Particle.Best.Cost = [];
20 Particle = repmat(Particle,nParticle,1);
21 gBest.Cost = -inf;
22 gBest.Position = zeros(varSize);
23 for i = 1 : nParticle
24     for j = 1 : nVar
25         Particle(i).Position(1,j) = unifrnd(varMin(j),varMax(j),1);
26     end
27     Particle(i).Cost = costFun(Particle(i).Position, outline_im1, outline_im2);
28     Particle(i).Velocity = zeros(varSize);
29     Particle(i).Best.Position = Particle(i).Position;
30     Particle(i).Best.Cost = Particle(i).Cost;
31     if Particle(i).Best.Cost > gBest.Cost
32         gBest.Cost = Particle(i).Best.Cost;
33         gBest.Position = Particle(i).Best.Position;
34     end
35 end
36 %% Main Loop
37 for j = 1 : maxIt
38     for i=1:nParticle
39         Particle(i).Velocity  = w*Particle(i).Velocity ...
40             + rand(varSize)*c1.*(Particle(i).Best.Position- Particle(i).Position)...
41             + rand(varSize)*c2.*(gBest.Position-Particle(i).Position);
42         Particle(i).Position = Particle(i).Position+Particle(i).Velocity;
43         id1= Particle(i).Position > varMax;
44         id2= Particle(i).Position < varMin;
45         Particle(i).Position(id1) = varMax(id1);
46         Particle(i).Cost = costFun(Particle(i).Position, outline_im1, outline_im2);
47         if Particle(i).Cost> Particle(i).Best.Cost
48             Particle(i).Best.Cost = Particle(i).Cost;
49             Particle(i).Best.Position = Particle(i).Position;
50         end
51         if Particle(i).Best.Cost > gBest.Cost
52             gBest.Cost = Particle(i).Best.Cost;
53             gBest.Position = Particle(i).Best.Position;
```

```
54          end
55      end
56      w = w*wDamp;
57  end
58  BestParticle = gBest.Position;
59  end
60  function y = costFun(indParticle,...
61     outline_im1,...
62     outline_im2)
63  Xc = indParticle(1);
64  Yc = indParticle(2);
65  Alpha = indParticle(3);
66  rotateMatrix = @(t) ([cosd(t) -sind(t) ; sind(t) cosd(t)]);
67  outline_im2_PSO = [outline_im2(:,1)-Xc     outline_im2(:,2)-Yc];
68  outline_im2_PSO = (rotateMatrix(-Alpha)*outline_im2_PSO')';
69  y = distanceFun(outline_im1,outline_im2_PSO);
70  x1 = outline_im2_PSO(:,1);
71  y1 = outline_im2_PSO(:,2);
72  teta2 = wingsAngle(x1,y1);
73  teta1 = wingsAngle(outline_im1(:,1),outline_im1(:,1));
74  if  abs(abs(teta1)-abs(teta2))>10
75      y= -inf;
76  end
77  end
78  function d = distanceFun(outline_im1,outline_im2_PSO)
79  x1 = outline_im1(:,1);
80  y1 =outline_im1(:,2);
81  x2 = outline_im2_PSO(:,1);
82  y2 = outline_im2_PSO(:,2);
83  [X1,X2] = meshgrid(x1,x2);
84  [Y1,Y2] = meshgrid(y1,y2);
85  d = -mean(min(sqrt(((X1-X2).^2)+((Y1-Y2).^2))));
86  end
87  function teta = wingsAngle(x,y)
88  d_old=0;
89  for i = 1 : length(x)
90      id1= i ;
91      [d, id2] = max(sqrt(  ((x(i) - x).^2) + ((y(i)-y).^2)  ));
92      if d>d_old
93          lineID = [id1 id2];
94      end
95  end
96  teta = atand((y(lineID(2))-y(lineID(1)))/(x(lineID(2))-x(lineID(1))));
97  end
```

## Code S12: Computing Asymmetry

This function is developed for computing the asymmetry of two wings.
**Inputs.** Code S12 requires 18 inputs.

- *image1_cellsCentroid*. An N-by-2 array contains the x and y coordinates of the cell centroid coordinates of the first wing. N refers to the number of cells in the first image.
- *image2_cellsCentroid*. An N-by-2 array contains the x and y coordinates of the cell centroid coordinates of the second wing. N refers to the number of cells in the second image.
- *image1_cellsArea*. An N-by-1 array containing the area of cells in the first image. N refers to the number of cells in the first image.
- *image2_cellsArea*. An N-by-1 array containing the area of cells in the second image. N refers to the number of cells in the second image.

- *image1_cellsLength*. An N-by-1 array containing the length of cells in the first image. N refers to the number of cells in the first image.
- *image2_cellsLength*. An N-by-1 array containing the length of cells in the second image. N refers to the number of cells in the second image.
- *image1_cellsWidth*. An N-by-1 array containing the width of cells in the first image. N refers to the number of cells in the first image.
- *image2_cellsWidth*. An N-by-1 array containing the width of cells in the second image. N refers to the number of cells in the second image.
- *image1_cellsCircularity*. An N-by-1 array containing the circularity of cells in the first image. N refers to the number of cells in the first image.
- *image2_cellsCircularity*. An N-by-1 array containing the circularity of cells in the second image. N refers to the number of cells in the second image.
- *image1_junctionsAmount*. Number of junctions in the first image.
- *image2_junctionsAmount*. Number of junctions in the second image.
- *image1_cellsAmount*. Number of cells in the first image.
- *image2_cellsAmount*. Number of cells in the second image.
- *image1_junctionsCoordinate*. An N-by-2 array containing the coordinate of junctions in the first image. N refers to the number of junctions in the first image.
- *image2_junctionsCoordinate*. An N-by-2 array containing the coordinate of junctions in the second image. N refers to the number of junctions in the second image.
- *image1_outline*. An N-by-2 array containing the coordinate of the first image outline. N refers to the number of points on the outline of the first image.
- *image2_outline*. An N-by-2 array containing the coordinate of the second image outline. N refers to the number of points on the outline of the second image.

**Outputs.** Code SCode S12 generates 16 outputs.

- *Im1VsIm2_cellsCentroidMeanDistance*. The mean distance between corresponding cell centroids of the wings.
- *Im1VsIm2_cellsCentroidMeanDistanceSTD*. The standard deviation of the mean distance between corresponding cell centroids of the wings.
- *Im1VsIm2_NRMSE_cellsArea*. The Normalized Root Mean Square Error of wing cells area.
- *Im1VsIm2_NRMSE_cellsLength*. The Normalized Root Mean Square Error of wing cells length.
- *Im1VsIm2_NRMSE_cellsWidth*. The Normalized Root Mean Square Error of wing cells width.
- *Im1VsIm2_NRMSE_cellsCircularity*. The Normalized Root Mean Square Error of wing cells circularity.

- *Im1AndIm2_junctionsAmountSubtract*. The difference in the number of junctions.
- *Im1AndIm2_cellsAmountSubtract*. The difference in the number of cells.
- *Im1VsIm2_junctionsMeanDistance*. The mean distance between the junctions of wings.
- *Im1VsIm2_junctionsMeanDistanceSTD*. The standard deviation of the mean distances between the junctions of wings.
- *Im1VsIm2_outlineMeanDistance*. the mean distance between the wings outlines.
- *Im1VsIm2_outlineMeanDistanceSTD*. The standard deviation of the mean distance between the wings outlines.
- *Im1VsIm2_Regression_cellsArea*. The regression between the area of the wing cells.
- *Im1VsIm2_Regression_cellsLength*. The regression between the length of the wing cells.
- *Im1VsIm2_Regression_cellsWidth*. The regression between the width of the wing cells.
- *Im1VsIm2_Regression_cellsCircularity*. The regression between the circularity of the wing cells.

**Code Descriptions.**

- **Lines 29-41.** The code in these lines finds the corresponding cells between the first and second images.
- **Line 42.** This line measures the mean distance of corresponding cells.
- **Line 43.** This line measures the standard deviation of the mean distance of corresponding cells.
- **Lines 45-48.** These lines compute the NRMSE between the corresponding cells' area, length, width, and circularity.
- **Line 50.** This line calculates the difference in the number of junctions between two wing images.
- **Line 51.** This line calculates the difference in the number of cells between two wing images.
- **Lines 53-62.** These lines calculate the mean distance of corresponding junctions and their standard deviations.
- **Lines 64-73.** These lines calculate the mean distance between wing outlines and their standard deviations.
- **Lines75-93.** These lines measure the regression of the area, length, width, and circularity of corresponding wing cells.

Code S12: The Developed code for computing the asymmetry of wings

```

function [Im1VsIm2_cellsCentroidMeanDistance,...
    Im1VsIm2_cellsCentroidMeanDistanceSTD,...
    Im1VsIm2_NRMSE_cellsArea,...
    Im1VsIm2_NRMSE_cellsLength,...
    Im1VsIm2_NRMSE_cellsWidth,...
    Im1VsIm2_NRMSE_cellsCircularity,...
    Im1AndIm2_junctionsAmountSubtract,...
    Im1AndIm2_cellsAmountSubtract,...
    Im1VsIm2_junctionsMeanDistance,...
    Im1VsIm2_junctionsMeanDistanceSTD,...
    Im1VsIm2_outlineMeanDistance,...
    Im1VsIm2_outlineMeanDistanceSTD,...
    Im1VsIm2_Regression_cellsArea,...
    Im1VsIm2_Regression_cellsLength,...
    Im1VsIm2_Regression_cellsWidth,...
    Im1VsIm2_Regression_cellsCircularity]=asymetryComputation(image1_cellsCentroid,...
    image2_cellsCentroid,image1_cellsArea,image2_cellsArea,image1_cellsLength,
        image2_cellsLength,...
    image1_cellsWidth,image2_cellsWidth,image1_cellsCircularity,image2_cellsCircularity,
        ...
    image1_junctionsAmount,image2_junctionsAmount,image1_cellsAmount,image2_cellsAmount,
        ...
    image1_junctionsCoordinate, image2_junctionsCoordinate,image1_outline,image2_outline
        )
% Asymmetry Computation: Image1 Vs Image2
Im1VsIm2_Area = zeros(image1_cellsAmount,2);
Im1VsIm2_Length = zeros(image1_cellsAmount,2);
Im1VsIm2_Width = zeros(image1_cellsAmount,2);
Im1VsIm2_Circularity = zeros(image1_cellsAmount,2);
Im1VsIm2_cellsCentroidDistance = zeros(image1_cellsAmount,1);
Im1VsIm2_correspondingCell = zeros(image1_cellsAmount,2);
for i = 1 : app.image1_cellsAmount
    x = image1_cellsCentroid(i,1);
    y = image1_cellsCentroid(i,2);
    d = sqrt(((x - image2_cellsCentroid(:,1)).^2)+((y - image2_cellsCentroid(:,2)).^2));
    [dMin,id] = min(d);
    j = id(1);
    Im1VsIm2_correspondingCell(i,:) =[i j];
    Im1VsIm2_Area(i,:) = [image1_cellsArea(i)      image2_cellsArea(j)];
    Im1VsIm2_Length(i,:) = [image1_cellsLength(i)    image2_cellsLength(j)];
    Im1VsIm2_Width(i,:) = [image1_cellsWidth(i)     image2_cellsWidth(j)];
    Im1VsIm2_Circularity(i,:) = [image1_cellsCircularity(i)     image2_cellsCircularity(j
        )];
    Im1VsIm2_cellsCentroidDistance(i) = dMin(1);
end
Im1VsIm2_cellsCentroidMeanDistance = mean(Im1VsIm2_cellsCentroidDistance);
Im1VsIm2_cellsCentroidMeanDistanceSTD = std(Im1VsIm2_cellsCentroidDistance);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im1VsIm2_NRMSE_cellsArea = goodnessOfFit(Im1VsIm2_Area(:,1),Im1VsIm2_Area(:,2),'NRMSE');
Im1VsIm2_NRMSE_cellsLength = goodnessOfFit(Im1VsIm2_Length(:,1),Im1VsIm2_Length(:,2),'
    NRMSE');
Im1VsIm2_NRMSE_cellsWidth = goodnessOfFit(Im1VsIm2_Width(:,1),Im1VsIm2_Width(:,2),'NRMSE
    ');
Im1VsIm2_NRMSE_cellsCircularity = goodnessOfFit(Im1VsIm2_Circularity(:,1),
    Im1VsIm2_Circularity(:,2),'NRMSE');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im1AndIm2_junctionsAmountSubtract = abs(image1_junctionsAmount -image2_junctionsAmount);
Im1AndIm2_cellsAmountSubtract = abs(image1_cellsAmount - image2_cellsAmount);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im1VsIm2_junctionsDistance = zeros(image1_junctionsAmount,1);
for i = 1:image1_junctionsAmount
    x = image1_junctionsCoordinate(i,1);
    y = image1_junctionsCoordinate(i,2);
    d = sqrt(((x - image2_junctionsCoordinate(:,1)).^2)+((y - image2_junctionsCoordinate
        (:,2)).^2));
    dMin = min(min(d));
    Im1VsIm2_junctionsDistance(i) = dMin;
end
Im1VsIm2_junctionsMeanDistance = mean(Im1VsIm2_junctionsDistance);
Im1VsIm2_junctionsMeanDistanceSTD = std(Im1VsIm2_junctionsDistance);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x = image1_outline(:,1);
y = image1_outline(:,2);
Im1VsIm2_outlineDistance = zeros(length(x),1);
for i = 1 : length(x)
    d = sqrt(((x(i) - image2_outline(:,1)).^2)+((y(i) - image2_outline(:,2)).^2));
    dMin = min(min(d));
    Im1VsIm2_outlineDistance(i) = dMin;
end
Im1VsIm2_outlineMeanDistance = mean(Im1VsIm2_outlineDistance);
Im1VsIm2_outlineMeanDistanceSTD = std(Im1VsIm2_outlineDistance);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mdl_area = fitlm(Im1VsIm2_Area(:,1),Im1VsIm2_Area(:,2));
Im1VsIm2_Reg_Area_x = mdl_area.Variables.x1;
Im1VsIm2_Reg_Area_y = mdl_area.Variables.y;
Im1VsIm2_Regression_cellsArea = regression(Im1VsIm2_Reg_Area_x' , Im1VsIm2_Reg_Area_y');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mdl_length = fitlm(Im1VsIm2_Length(:,1),Im1VsIm2_Length(:,2));
Im1VsIm2_Reg_Length_x = mdl_length.Variables.x1;
Im1VsIm2_Reg_Length_y = mdl_length.Variables.y;
Im1VsIm2_Regression_cellsLength = regression(Im1VsIm2_Reg_Length_x',
    Im1VsIm2_Reg_Length_y');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mdl_width = fitlm(Im1VsIm2_Width(:,1),Im1VsIm2_Width(:,2));
Im1VsIm2_Reg_Width_x = mdl_width.Variables.x1;
Im1VsIm2_Reg_Width_y = mdl_width.Variables.y;
Im1VsIm2_Regression_cellsWidth = regression(Im1VsIm2_Reg_Width_x',Im1VsIm2_Reg_Width_y')
    ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mdl_circularity = fitlm(Im1VsIm2_Circularity(:,1),Im1VsIm2_Circularity(:,2));
Im1VsIm2_Reg_Circularity_x = mdl_circularity.Variables.x1;
Im1VsIm2_Reg_Circularity_y = mdl_circularity.Variables.y;
Im1VsIm2_Regression_cellsCircularity = regression(Im1VsIm2_Reg_Circularity_x',
    Im1VsIm2_Reg_Circularity_y');
end
```

## Code S13: Ramer-Douglas-Peucker Line Simplification

Code S13 is the function developed for Line Simplification. Line 1 in Code S13 defines the function.

**Input:**

- $X$: X coordinate of the input curve.
- $Y$: Y coordinate of the input curve.
- *epsilon*: The accuracy of the output curve

**Ouput:** The outputs of this function are as follows:

- *xSMP*: X coordinate of simplified curve.
- *ySMP*: y coordinate of simplified curve.

Code S13: Ramer-Douglas-Peucker Line Simplification code in MATLAB

```
function [xSMP,ySMP]=RDP_lineSimplification(X,Y,epsilon)
close all
clc
global allPoints
id= isnan(Y);
X(id) = [];
Y(id) = [];
H = zeros(1,length(X));
```

```
for i = 1 : length(X)
    H(i)=findHight([X(1) X(i) X(end)],[Y(1) Y(i) Y(end)]);
end
epsilon =epsilon*(max(max(H)));
allPoints = [X' Y'];
plot(X,Y,'b-','linewidth',2)
hold on
fPid = [1 length(X)]; %Final Points ID
n = 2;
e=1;
while e~=0
    newPoints = [];
    for i=1:length(fPid)-1
        hID=findPoint(fPid(i),fPid(i+1),epsilon);
        newPoints =[newPoints hID]; %#ok
    end
    fPid = [fPid newPoints];     %#ok
    fPid = sort(fPid);
    e = abs(length(fPid)-n);
    n = length(fPid);
end
plot(X(fPid),Y(fPid),'ro','linewidth',2,'markersize',10)
xSMP = X(fPid);
ySMP = Y(fPid);
end
function hID=findPoint(idP1,idP2,epsilon)
global allPoints
CPs = idP1+1 : idP2-1;
H = zeros(1,length(CPs));
for i = 1 : length(CPs)
    H(i)=findHight([allPoints(idP1,1) allPoints(CPs(i),1) allPoints(idP2,1)],[allPoints(
        idP1,2) allPoints(CPs(i),2) allPoints(idP2,2)]);
end
if ~isempty(H) && max(max(H))>epsilon
    [~,id] = max(H);
    hID = CPs(id(1));
else
    hID = [];
end
end
function H=findHight(X,Y)
% X is a 1 by 3 matrix
% Y is a 1 by 3 matrix
X = [X X(1)];
Y = [Y Y(1)];
A = (sum(X(1:end-1).*Y(2:end))-sum(X(2:end).*Y(1:end-1)))/2;
d = sqrt(((X(3)-X(1))^2)+((Y(3)-Y(1))^2));
H = 2*A/d;
H = abs(H);
end
```

## Code S14: Assignment of Corrugation, Using Secondary Image

Code S14: Corrugation assignment code in MATLAB

```
function [GeometryData,remove_face,refPoint,refL,refU] = corrugationAssignment(
    numberOfSections,Height,nSharpen, boundary )
[FileName,PathName,FilterIndex] = uigetfile('*','Select an Image');
if FilterIndex==0
    errordlg('An image is required')
    return
end
w=imread([PathName FileName]);
if numel(size(w))==3
    w=rgb2gray(w);
end
s_a= size(w);
id1 = w>245;
id2 = w<10 ;
```

```
w(id1) = 255;
w(id2) = 0;
w=(double(w)/255);
u=4:size(w,2)-4;
u1=4:size(w,1)-4;
wAll = cell(200,1);
for i1=1:200
    w(u1,u)=(10*w(u1,u)+w(u1,u+1)+w(u1,u+2)+w(u1,u+3)+w(u1,u-1)+w(u1,u-2)+w(u1,u-3)...
        +w(u1-1,u)+w(u1-2,u)+w(u1-3,u)+w(u1+1,u)+w(u1+2,u)+w(u1+3,u))/22;
    w = w-min(min(w));
    w = w/max(max(w));
    w= round(w*10000);
    w = w/10000;
    wAll{i1} = w;
end
w=wAll{nSharpen};
w=w*Height;
w=w(end:-1:1,:);
Frame = [min(min(boundary(:,1)))+2 max(max(boundary(:,1)))-2 min(min(boundary(:,2))) max
    (max(boundary(:,2)))];
Frame=round(Frame);
columns=round(linspace(Frame(1),Frame(2),numberOfSections));
for i=1:numberOfSections
    first_y = 1;
    last_y  = size(w,1);
    h=w(first_y:last_y,columns(i))';
    lat = (first_y:last_y)';
    lon = h';
    [latreduced, lonreduced] = reducem(lat, lon,.7);
    l=length(latreduced);
    Coordinates=[ones(1,l)*columns(i);latreduced';lonreduced'];
    laftData{i,1} = Coordinates;
    laftData{i,2} = HeightKnob;  % The value of Height for the section
    laftData{i,3} = nSharpen;   % The value of Sharpness for the section
end
remove_face=[ones(1,length(columns)-1)*s_a(1) ones(1,length(columns)-1);...
    zeros(1,2*length(columns)-2)
    (columns(1:end-1)+columns(2:end))/2 (columns(1:end-1)+columns(2:end))/2];
Geometry_Data=cell(3,numberOfSections);
origin_x = 0;
origin_y = 0;
for i=1:numberOfSections
    Geometry_Data{3,i} = [origin_x origin_y];
    X = abs(laftData{i,1}(2,:));
    Y = abs(laftData{i,1}(3,:));
    origin_x = 0;
    origin_y = 0;
    Distance = laftData{i}(1,1)-laftData{1}(1,1);
    Geometry_Data{1,i} = [X;Y];
    Geometry_Data{2,i} = Distance;
end
refPoint=500;
refL=[500 500 1;500 500 2];
refU=[500 1 1;500 2 1];
remove_face(3,:)=remove_face(3,:)-Frame(1);
end
```

---

## Code S15: Generating 3D Corrugated JNL File

Code S15: Generating JNL file for 3D corrugated model

```
function JNL3D(numberOfSections,refPoint,refL,refU,GeometryData,remove_face,Domains,
    Discontinuity,laftData)
Geometry_Data= GeometryData;
Domains3D=[];
a=[];
for i=1:numberOfSections
    b=Geometry_Data{1,i};
    d=length(b);
```

```
    a=[a ; d];  %#ok
end
MAX_LAFT=(max(a));
for i=1:size(Domains,1)
    A=[Domains{i,2}(:,1)-round((Geometry_Data{2,end})/2)-laftData{1}(1,1),...
        Domains{i,2}(:,2)-round(MAX_LAFT/2)];
    Domains3D{end+1}=A;
end
if app.disExistence~=1
    app.Discontinuity3D = 0;
else
    for i=1:size(Discontinuity,1)
        A=[Discontinuity{i,2}(:,1)-round((Geometry_Data{2,end})/2)-laftData{1}(1,1),...
            Discontinuity{i,2}(:,2)-round(MAX_LAFT/2)];
        Discontinuity3D{i}=A;
    end
end
final_section = Domains3D;
if ~isempty(Discontinuity3D)
    hole = Discontinuity3D;
else
    hole = [];
end
Reference.l= refL;
Reference.u= refU;
Reference.point= refPoint;
hole_existence= 1;
a=[];
for i=1:numberOfSections
    b=Geometry_Data{1,i};
    d=length(b);
    a=[a ; d];
end
MAX_LAFT=(max(a))/2;
NMNM=7+numberOfSections+1;
[file,path] = uiputfile('Corrugated 3D Shell.py','Save as');
fid=fopen([path file],'w');
fprintf(fid,'# -*- coding: mbcs -*- \n');
fprintf(fid,'from part import * \n');
fprintf(fid,'from material import * \n');
fprintf(fid,'from section import * \n');
fprintf(fid,'from assembly import * \n');
fprintf(fid,'from step import * \n');
fprintf(fid,'from interaction import * \n');
fprintf(fid,'from load import * \n');
fprintf(fid,'from mesh import * \n');
fprintf(fid,'from optimization import * \n');
fprintf(fid,'from job import * \n');
fprintf(fid,'from sketch import * \n');
fprintf(fid,'from visualization import * \n');
fprintf(fid,'from connectorBehavior import * \n');
fprintf(fid,'mdb.models[''Model-1''].Part(dimensionality=THREE_D, name=''Part-1'', type=
    DEFORMABLE_BODY) \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].ReferencePoint(point=(0,0,0)) \n'
    );
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,%g,%g)) \n',Reference.u(1,:));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,%g,%g)) \n',Reference.u(2,:));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,%g,%g)) \n',Reference.l(1,:));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,%g,%g)) \n',Reference.l(2,:));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumAxisByTwoPoint(point1= \n');
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].datums[2], point2= \n');
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].datums[3]) \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumAxisByTwoPoint(point1= \n');
fprintf(fid,' mdb.models[''Model-1''].parts[''Part-1''].datums[4], point2= \n');
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].datums[5]) \n');
for i=1:numberOfSections
    fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPlaneByPrincipalPlane(
        offset=%g, principalPlane=XYPLANE) \n',Geometry_Data{2,i});
end
```

```
74  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPlaneByPrincipalPlane(offset
        =%g, principalPlane=XZPLANE) \n',Reference.point);
75  for i=1:numberOfSections
76      fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(gridSpacing=75, name=''
            __profile__'', sheetSize=3000, transform= \n');
77      fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].MakeSketchTransform( \n');
78      fprintf(fid,'sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g], \n',i
            +7);
79      fprintf(fid,'sketchPlaneSide=SIDE1, \n');
80      fprintf(fid,' sketchUpEdge=mdb.models[''Model-1''].parts[''Part-1''].datums[6], \n'
            );
81      fprintf(fid,'sketchOrientation=RIGHT, origin=(%g,%g,%g))) \n',Geometry_Data{3,i}(1),
            Geometry_Data{3,i}(2),Geometry_Data{2,i});
82      fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].projectReferencesOntoSketch(
            filter= \n');
83      fprintf(fid,' COPLANAR_EDGES, sketch=mdb.models[''Model-1''].sketches[''__profile__
            '']) \n');
84      fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Spline(points=( \n');
85      for j=1:length(Geometry_Data{1,i})-1
86          fprintf(fid,'(%g , %g), ',Geometry_Data{1,i}(1,j),Geometry_Data{1,i}(end,j));
87      end
88      fprintf(fid,'(%g,%g))) \n',Geometry_Data{1,i}(1,end),Geometry_Data{1,i}(end,end));
89      fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Wire(sketch= \n');
90      fprintf(fid,' mdb.models[''Model-1''].sketches[''__profile__''], sketchOrientation=
            RIGHT, \n');
91      fprintf(fid,'     sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g], \
            n',i+7);
92      fprintf(fid,'     sketchPlaneSide=SIDE1, sketchUpEdge= \n');
93      fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].datums[6]) \n');
94      fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
95  end
96  for i=1:numberOfSections-1
97      fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].ShellLoft(endCondition=NONE,
            \n');
98      fprintf(fid,'     loftsections=((mdb.models[''Model-1''].parts[''Part-1''].edges.
            findAt((%g,%g,%g), ), ), ( \n',Geometry_Data{1,i+1}(1,2),Geometry_Data{1,i
            +1}(2,2),Geometry_Data{2,i+1});
99      fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].edges.findAt((%g,%g,%g),
            ), )), startCondition=NONE) \n',Geometry_Data{1,i}(1,2),Geometry_Data{1,i}(2,2),
            Geometry_Data{2,i});
100 end
101 fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(gridSpacing=75, name=''
        __profile__'', sheetSize=3000, transform= \n');
102 fprintf(fid,'  mdb.models[''Model-1''].parts[''Part-1''].MakeSketchTransform( \n');
103 fprintf(fid,' \n');
104 fprintf(fid,'sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g], \n',NMNM)
        ;
105 fprintf(fid,'  sketchPlaneSide=SIDE1, \n');
106 fprintf(fid,'  sketchUpEdge=mdb.models[''Model-1''].parts[''Part-1''].datums[7], \n');
107 fprintf(fid,'sketchOrientation=TOP, origin=(%g,%g,%g))) \n',MAX_LAFT,Reference.point,(
        Geometry_Data{2,end})/2);
108 fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].projectReferencesOntoSketch(
        filter= \n');
109 fprintf(fid,' COPLANAR_EDGES, sketch=mdb.models[''Model-1''].sketches[''__profile__''])
        \n');
110 for i=1:length(final_section)
111     for j=1:length(final_section{i})-1
112         fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g ,
                %g), point2=(%g , %g)) \n',final_section{i}(j,1),final_section{i}(j,2),
                final_section{i}(j+1,1),final_section{i}(j+1,2));
113     end
114     fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g , %g)
            , point2=(%g , %g)) \n',final_section{i}(end-1,1),final_section{i}(end-1,2),
            final_section{i}(end,1),final_section{i}(end,2));
115 end
116 fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].PartitionFaceBySketchThruAll(
        faces= \n');
117 fprintf(fid,'  mdb.models[''Model-1''].parts[''Part-1''].faces.findAt( \n');
118 for i=1:numberOfSections
119     fprintf(fid,'((%g,%g,%g), ), ',Geometry_Data{1,i}(1,1),Geometry_Data{1,i}(end,1),
            Geometry_Data{2,i});
120 end
121 fprintf(fid,'   ), sketch= \n');
```

```
122 fprintf(fid,'    mdb.models[''Model-1''].sketches[''__profile__''], sketchOrientation=TOP
        , \n');
123 fprintf(fid,'sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g], \n',NMNM)
        ;
124 fprintf(fid,'     sketchPlaneSide=SIDE1, sketchUpEdge= \n');
125 fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].datums[7]) \n');
126 fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
127 if hole_existence==1
128     fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(gridSpacing=75, name=''
            __profile__'', sheetSize=3000, transform= \n');
129     fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].MakeSketchTransform( \n');
130     fprintf(fid,'    sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g], \n
            ',NMNM);
131     fprintf(fid,'     sketchPlaneSide=SIDE1,  \n');
132     fprintf(fid,'  sketchUpEdge=mdb.models[''Model-1''].parts[''Part-1''].datums[7],  \n
            ');
133     fprintf(fid,'    sketchOrientation=TOP, origin=(%g,%g,%g))) \n',MAX_LAFT,Reference.
            point,(Geometry_Data{2,end})/2);
134     fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].projectReferencesOntoSketch(
            filter= \n');
135     fprintf(fid,'  COPLANAR_EDGES, sketch=mdb.models[''Model-1''].sketches[''__profile__
            '']) \n');
136     for i=1:length(hole)
137         for j=1:length(hole{i})-1
138             fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g, %g),point2=(%g,%g)) \n',hole{i}(j,1),hole{i}(j,2),hole{i}(j+1,1),
                hole{i}(j+1,2));
139         end
140     end
141     fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].CutExtrude(
            flipExtrudeDirection=OFF,  \n');
142     fprintf(fid,'  sketch=mdb.models[''Model-1''].sketches[''__profile__''],
            sketchOrientation= \n');
143     fprintf(fid,'    TOP, sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].datums[%g
            ], \n',NMNM);
144     fprintf(fid,'     sketchPlaneSide=SIDE1, sketchUpEdge= \n');
145     fprintf(fid,'  mdb.models[''Model-1''].parts[''Part-1''].datums[7]) \n');
146     fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
147     fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=False
            , faceList=( \n');
148     for i=1:length(remove_face)-1
149         fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].faces.findAt( \n');
150         fprintf(fid,'    ( %g , %g , %g ), ), \n',remove_face(1,i),Geometry_Data{1,1}(end
                ,1),remove_face(3,i));
151     end
152     fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g , %g , %g )
            ,))) ',remove_face(1,end),Geometry_Data{1,1}(end,1),remove_face(3,end));
153     fclose(fid);
154 else
155     fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=False
            , faceList=( \n');
156     for i=1:length(remove_face)-1
157         fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].faces.findAt( \n');
158         fprintf(fid,'    ( %g , %g , %g ), ), \n',remove_face(1,i),Geometry_Data{1,1}(end
                ,1),remove_face(3,i));
159     end
160     fprintf(fid,'     mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g , %g , %
            g ),))) ',remove_face(1,end),Geometry_Data{1,1}(end,1),remove_face(3,end));
161     fclose(fid);
162 end
163 end
```

## Code S16: Generating 2D Shell JNL File, with two sections

Code S16: Generating JNL file for 3D corrugated model

```
1
2 function JNL_Planar_Double_Section2D(Domains,imageSize)
3 part = cell(1,size(Domains,1));
```

```
for i=1:size(Domains,1)
    part{i} = Domains{i,2};
end
Dis = cell(1,size(Discontinuity,1));
for i=1:size(Discontinuity,1)
    Dis{i}  = Discontinuity{i,2};
end
SS=(max(imageSize))*2;
[file,path] = uiputfile('Flat 2DPlanar Two-Section.py','Save as');
fid=fopen([path file],'w');
fprintf(fid,'from part import * \n');
fprintf(fid,'from material import * \n');
fprintf(fid,'from section import * \n');
fprintf(fid,'from assembly import * \n');
fprintf(fid,'from step import * \n');
fprintf(fid,'from interaction import * \n');
fprintf(fid,'from load import * \n');
fprintf(fid,'from mesh import * \n');
fprintf(fid,'from optimization import * \n');
fprintf(fid,'from job import * \n');
fprintf(fid,'from sketch import * \n');
fprintf(fid,'from visualization import * \n');
fprintf(fid,'from connectorBehavior import * \n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
    g)\n',SS);
for i=1:length(part{1})-1
    fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g, %g),
        point2=(%g, %g))\n',part{1}(i,1),part{1}(i,2),part{1}(i+1,1),part{1}(i+1,2));
end
fprintf(fid,'mdb.models[''Model-1''].Part(dimensionality=TWO_D_PLANAR, name=''Part-1'',
    type=DEFORMABLE_BODY) \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].BaseShell(sketch=mdb.models[''
    Model-1''].sketches[''__profile__''])\n');
fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(gridSpacing=4.88, name=''
    __profile__'', \n');
fprintf(fid,'    sheetSize=%g, transform=\n',SS);
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].MakeSketchTransform(\n');
fprintf(fid,'    sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g,
    %g, 0.0), (0.0, 0.0, 1.0)), sketchPlaneSide=SIDE1, \n',part{1}(1,1),part{1}(1,2));
fprintf(fid,'    sketchOrientation=RIGHT, origin=(0, 0, 0.0)))\n');
if length(part)>2
    for j=2:length(part)
        for i=1:length(part{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g , %g), point2=(%g , %g)) \n',part{j}(i,1),part{j}(i,2),part{j}(i
                +1,1),part{j}(i+1,2));
        end
    end
end
if ~isempty(Dis)
    for j=1:length(Dis)
        for i=1:length(Dis{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g, %g), point2=(%g, %g)) \n',Dis{j}(i,1),Dis{j}(i,2),Dis{j}(i+1,1),
                Dis{j}(i+1,2));
        end
    end
end
if length(part)>2
    fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].PartitionFaceBySketch(faces=\
        n');
    fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt(((%g, %g,
        0.0), )),\n',part{1}(1,1),part{1}(1,2));
    fprintf(fid,'    sketch=mdb.models[''Model-1''].sketches[''__profile__''])\n');
    fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
end
if ~isempty(Dis)
    for i=1:length(Dis)
        in = inpolygon(Discontinuity{i,4}(:,1),app.imageSize(1)-Discontinuity{i,4}(:,2),
            Discontinuity{i,2}(:,1),Discontinuity{i,2}(:,2));
        id = find(in==1);
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=
```

```
            False, faceList=(\n');
64          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g, %g,
                0.0), ), ))\n', Discontinuity{i,4}(id(1),1),app.imageSize(1)-Discontinuity{
            i,4}(id(1),2));
65      end
66  end
67  fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-1'')\n');
68  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-1''].Density(table=((1, ), ))\
    n');
69  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-1''].Elastic(table=((1, 1), ))
    \n');
70  fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-2'')\n');
71  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-2''].Density(table=((1, ), ))\
    n');
72  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-2''].Elastic(table=((1, 1), ))
    \n');
73  for j=1:size(Domains,1)
74      if j==1
75          fprintf(fid,'mdb.models[''Model-1''].HomogeneousSolidSection(material=''Material
                -1'', name=''Section-1'', thickness=None)\n');
76          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0), )), name=''Set-1''
                )\n',Domains{j,2}(1,1),Domains{j,2}(1,2));
77          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
78          fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
79          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-1''],
                sectionName=\n');
80          fprintf(fid,'    ''Section-1'', thicknessAssignment=FROM_SECTION)\n');
81      else
82          in = inpolygon(Domains{j,4}(:,1),app.imageSize(1)-Domains{j,4}(:,2),Domains{j
                ,2}(:,1),Domains{j,2}(:,2));
83          id = find(in==1);
84          fprintf(fid,'mdb.models[''Model-1''].HomogeneousSolidSection(material=''Material
                -2'', name=''Section-2'', thickness=None)\n');
85          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0), )), name=''Set-%g'
                ')\n',Domains{j,4}(id(1),1),app.imageSize(1)-Domains{j,4}(id(1),2),j);
86          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
87          fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
88          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-%g''],
                sectionName=\n',j);
89          fprintf(fid,'    ''Section-2'', thicknessAssignment=FROM_SECTION)\n');
90      end
91  end
92  fclose(fid);
93  end
```

---

## Code S17: Generating 3D Shell JNL File, with two sections

Code S17: Generating JNL file for 3D shell with double sections

```
1   function JNL_Planar_Double_Section3D(Domains,Discontinuity,imageSize)
2   id = [];
3   for j=2:size(Domains,1)
4       try
5           in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
                ,2}(:,1),Domains{j,2}(:,2));
6           if sum(in)==0
7               id=[id;j];
8           end
9       catch
10          id=[id;j];
11      end
12  end
13  part = cell(1,size(Domains,1));
14  for i=1:size(Domains,1)
15      part{i} = Domains{i,2};
```

```
end
Dis = cell(1,size(Discontinuity,1));
for i=1:size(Discontinuity,1)
    Dis{i} = Discontinuity{i,2};
end
SS=(max(imageSize))*2;
[file,path] = uiputfile('Flat 3DShell Two-Section.py','Save as');
fid=fopen([path file],'w');
fprintf(fid,'from part import * \n');
fprintf(fid,'from material import * \n');
fprintf(fid,'from section import * \n');
fprintf(fid,'from assembly import * \n');
fprintf(fid,'from step import * \n');
fprintf(fid,'from interaction import * \n');
fprintf(fid,'from load import * \n');
fprintf(fid,'from mesh import * \n');
fprintf(fid,'from optimization import * \n');
fprintf(fid,'from job import * \n');
fprintf(fid,'from sketch import * \n');
fprintf(fid,'from visualization import * \n');
fprintf(fid,'from connectorBehavior import * \n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
    g)\n',SS);
for i=1:length(part{1})-1
    fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g, %g),
        point2=(%g, %g))\n',part{1}(i,1),part{1}(i,2),part{1}(i+1,1),part{1}(i+1,2));
end
fprintf(fid,'mdb.models[''Model-1''].Part(dimensionality=THREE_D, name=''Part-1'', type=
    DEFORMABLE_BODY) \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].BaseShell(sketch=mdb.models[''
    Model-1''].sketches[''__profile__''])\n');
fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,0.0, 0.0))\n',imageSize(2));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
    ,20.0, 0.0))\n',imageSize(2));
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumAxisByTwoPoint(point1=mdb.
    models[''Model-1''].parts[''Part-1''].datums[3], point2=mdb.models[''Model-1''].
    parts[''Part-1''].datums[2])\n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPlaneByPrincipalPlane(offset
    =0.0,principalPlane=XYPLANE)\n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
    g)\n',SS);
if length(part)>2
    for j=2:length(part)
        for i=1:length(part{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g , %g), point2=(%g , %g)) \n',part{j}(i,1),part{j}(i,2),part{j}(i
                +1,1),part{j}(i+1,2));
        end
    end
end
if ~isempty(Dis)
    for j=1:length(Dis)
        for i=1:length(Dis{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g, %g), point2=(%g, %g)) \n',Dis{j}(i,1),Dis{j}(i,2),Dis{j}(i+1,1),
                Dis{j}(i+1,2));
        end
    end
end
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].PartitionFaceBySketch(faces=\n');
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0),
    )),\n',part{1}(1,1),part{1}(1,2)); %?
fprintf(fid,'    sketch=mdb.models[''Model-1''].sketches[''__profile__''], \n');
fprintf(fid,'    sketchUpEdge=mdb.models[''Model-1''].parts[''Part-1''].datums[4])\n');
fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
if ~isempty(Dis)
    for i=1:length(Dis)
        in = inpolygon(Discontinuity{i,4}(:,1),imageSize(1)-Discontinuity{i,4}(:,2),
            Discontinuity{i,2}(:,1),Discontinuity{i,2}(:,2));
        id = find(in==1);
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=
```

```
            False, faceList=(\n');
73          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g, %g,
                0.0), ), ))\n', Discontinuity{i,4}(id(1),1),imageSize(1)-Discontinuity{i
                ,4}(id(1),2));
74      end
75  end
76  fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-1'')\n');
77  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-1''].Density(table=((1, ), ))\
        n');
78  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-1''].Elastic(table=((1, 1), ))
        \n');
79  fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-2'')\n');
80  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-2''].Density(table=((1, ), ))\
        n');
81  fprintf(fid,'mdb.models[''Model-1''].materials[''Material-2''].Elastic(table=((1, 1), ))
        \n');
82  fprintf(fid,'mdb.models[''Model-1''].HomogeneousShellSection(idealization=
        NO_IDEALIZATION, \n');
83  fprintf(fid,'    integrationRule=SIMPSON, material=''Material-1'', name=''Section-1'', \
        n');
84  fprintf(fid,'    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF, temperature=\
        n');
85  fprintf(fid,'    GRADIENT, thickness=1000.0, thicknessField='''', thicknessModulus=None,
         \n');
86  fprintf(fid,'    thicknessType=UNIFORM, useDensity=OFF)\n');
87  fprintf(fid,'mdb.models[''Model-1''].HomogeneousShellSection(idealization=
        NO_IDEALIZATION, \n');
88  fprintf(fid,'    integrationRule=SIMPSON, material=''Material-2'', name=''Section-2'', \
        n');
89  fprintf(fid,'    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF, temperature=\
        n');
90  fprintf(fid,'    GRADIENT, thickness=1000.0, thicknessField='''', thicknessModulus=None,
         \n');
91  fprintf(fid,'    thicknessType=UNIFORM, useDensity=OFF)\n');
92  for j=1:size(Domains,1)
93      if j==1
94          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g , %g , 0.0), )), name=''Set-1
                '') \n',Domains{j,2}(1,1),Domains{j,2}(1,2));
95          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
96          fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
97          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-1''],
                sectionName=\n');
98          fprintf(fid,'    ''Section-1'', thicknessAssignment=FROM_SECTION)\n');
99      else
100         in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
                ,2}(:,1),Domains{j,2}(:,2));
101         id = find(in==1);
102         fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g , %g , 0.0), )), name=''Set-%
                g'') \n',Domains{j,4}(id(1),1),imageSize(1)-Domains{j,4}(id(1),2),j);
103         fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
104         fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
105         fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-%g''],
                sectionName=\n',j);
106         fprintf(fid,'    ''Section-2'', thicknessAssignment=FROM_SECTION)\n');
107     end
108 end
109 fclose(fid);
110 end
```

## Code S18: Generating 2D Shell JNL File, With Multi Sections

Code S18: Generating JNL file for 2D shell with multi sections

```
1   function JNL_Planar_Multi_Section2D(Domains,Discontinuity,imageSize)
2   id = [];
```

```
for j=2:size(Domains,1)
    try
        in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
            ,2}(:,1),Domains{j,2}(:,2));
        if sum(in)==0
            id=[id;j];
        end
    catch
        id=[id;j];
    end
end
part = cell(1,size(Domains,1));
for i=1:size(Domains,1)
    part{i} = Domains{i,2};
end
Dis = cell(1,size(Discontinuity,1));
for i=1:size(Discontinuity,1)
    Dis{i}  = Discontinuity{i,2};
end
SS=(max(imageSize))*2;
[file,path] = uiputfile('Flat 2DPlanar Multi-Section.py','Save as');
fid=fopen([path file],'w');
fprintf(fid,'from part import * \n');
fprintf(fid,'from material import * \n');
fprintf(fid,'from section import * \n');
fprintf(fid,'from assembly import * \n');
fprintf(fid,'from step import * \n');
fprintf(fid,'from interaction import * \n');
fprintf(fid,'from load import * \n');
fprintf(fid,'from mesh import * \n');
fprintf(fid,'from optimization import * \n');
fprintf(fid,'from job import * \n');
fprintf(fid,'from sketch import * \n');
fprintf(fid,'from visualization import * \n');
fprintf(fid,'from connectorBehavior import * \n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
    g)\n',SS);
for i=1:length(part{1})-1
    fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g, %g),
        point2=(%g, %g))\n',part{1}(i,1),part{1}(i,2),part{1}(i+1,1),part{1}(i+1,2));
end
fprintf(fid,'mdb.models[''Model-1''].Part(dimensionality=TWO_D_PLANAR, name=''Part-1'',
    type=DEFORMABLE_BODY) \n');
fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].BaseShell(sketch=mdb.models[''
    Model-1''].sketches[''__profile__''])\n');
fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(gridSpacing=4.88, name=''
    __profile__'', \n');
fprintf(fid,'    sheetSize=%g, transform=\n',SS);
fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].MakeSketchTransform(\n');
fprintf(fid,'    sketchPlane=mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g,
    %g, 0.0), (0.0, 0.0, 1.0)), sketchPlaneSide=SIDE1, \n',part{1}(1,1),part{1}(1,2));
fprintf(fid,'    sketchOrientation=RIGHT, origin=(0, 0, 0.0)))\n');
if length(part)>2
    for j=2:length(part)
        for i=1:length(part{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g , %g), point2=(%g , %g)) \n',part{j}(i,1),part{j}(i,2),part{j}(i
                +1,1),part{j}(i+1,2));
        end
    end
end
if ~isempty(Dis)
    for j=1:length(Dis)
        for i=1:length(Dis{j})-1
            fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                =(%g, %g), point2=(%g, %g)) \n',Dis{j}(i,1),Dis{j}(i,2),Dis{j}(i+1,1),
                Dis{j}(i+1,2));
        end
    end
end
if length(part)>2
    fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].PartitionFaceBySketch(faces=\
```

```
        n');
65      fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt(((%g, %g,
            0.0), )),\n',part{1}(1,1),part{1}(1,2));
66      fprintf(fid,'    sketch=mdb.models[''Model-1''].sketches[''__profile__''])\n');
67      fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
68  end
69  if ~isempty(Dis)
70      for i=1:length(Dis)
71          in = inpolygon(Discontinuity{i,4}(:,1),imageSize(1)-Discontinuity{i,4}(:,2),
                Discontinuity{i,2}(:,1),Discontinuity{i,2}(:,2));
72          id = find(in==1);
73          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=
                False, faceList=(\n');
74          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g, %g,
                 0.0), ), ))\n', Discontinuity{i,4}(id(1),1),imageSize(1)-Discontinuity{i
                ,4}(id(1),2));
75      end
76  end
77  for j=1:size(Domains,1)
78      fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-%g'')\n',j);
79      fprintf(fid,'mdb.models[''Model-1''].materials[''Material-%g''].Density(table=((1, )
            , ))\n',j);
80      fprintf(fid,'mdb.models[''Model-1''].materials[''Material-%g''].Elastic(table=((1,
            1), ))\n',j);
81  end
82  for j=1:size(Domains,1)
83      if j==1
84          fprintf(fid,'mdb.models[''Model-1''].HomogeneousSolidSection(material=''Material
                -1'', name=''Section-1'', thickness=None)\n');
85          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0), )), name=''Set-1''
                )\n',Domains{j,2}(1,1),Domains{j,2}(1,2));
86          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
87          fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
88          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-1''],
                sectionName=\n');
89          fprintf(fid,'    ''Section-1'', thicknessAssignment=FROM_SECTION)\n');
90      else
91          in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
                ,2}(:,1),Domains{j,2}(:,2));
92          id = find(in==1);
93          fprintf(fid,'mdb.models[''Model-1''].HomogeneousSolidSection(material=''Material
                -%g'', name=''Section-%g'', thickness=None)\n',j,j);
94          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
                Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0), )), name=''Set-%g'
                ')\n',Domains{j,4}(id(1),1),imageSize(1)-Domains{j,4}(id(1),2),j);
95          fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
                =0.0, \n');
96          fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
97          fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-%g''],
                sectionName=\n',j);
98          fprintf(fid,'    ''Section-%g'', thicknessAssignment=FROM_SECTION)\n',j);
99      end
100 end
101 fclose(fid);
102 end
```

## Code S19: Generating JNL File for 3D Shell Models With Multi Sections

Code S19: Generating JNL file for 3D shell with multi sections

```
1   function JNL_Planar_Multi_Section3D(Domains,Discontinuity,imageSize)
2   id = [];
3   for j=2:size(Domains,1)
4       try
5           in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
                ,2}(:,1),Domains{j,2}(:,2));
```

```
6           if sum(in)==0
7               id=[id;j];
8           end
9       catch
10          id=[id;j];
11      end
12  end
13  part = cell(1,size(Domains,1));
14  for i=1:size(Domains,1)
15      part{i} = Domains{i,2};
16  end
17  Dis = cell(1,size(Discontinuity,1));
18  for i=1:size(Discontinuity,1)
19      Dis{i}  = Discontinuity{i,2};
20  end
21  SS=(max(imageSize))*2;
22  [file,path] = uiputfile('Flat 3DShell Multi-Section.py','Save as');
23  fid=fopen([path file],'w');
24  fprintf(fid,'from part import * \n');
25  fprintf(fid,'from material import * \n');
26  fprintf(fid,'from section import * \n');
27  fprintf(fid,'from assembly import * \n');
28  fprintf(fid,'from step import * \n');
29  fprintf(fid,'from interaction import * \n');
30  fprintf(fid,'from load import * \n');
31  fprintf(fid,'from mesh import * \n');
32  fprintf(fid,'from optimization import * \n');
33  fprintf(fid,'from job import * \n');
34  fprintf(fid,'from sketch import * \n');
35  fprintf(fid,'from visualization import * \n');
36  fprintf(fid,'from connectorBehavior import * \n');
37  fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
        g)\n',SS);
38  for i=1:length(part{1})-1
39      fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1=(%g, %g),
            point2=(%g, %g))\n',part{1}(i,1),part{1}(i,2),part{1}(i+1,1),part{1}(i+1,2));
40  end
41  fprintf(fid,'mdb.models[''Model-1''].Part(dimensionality=THREE_D, name=''Part-1'', type=
        DEFORMABLE_BODY) \n');
42  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].BaseShell(sketch=mdb.models[''
        Model-1''].sketches[''__profile__''])\n');
43  fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
44  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
        ,0.0, 0.0))\n',imageSize(2));
45  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPointByCoordinate(coords=(%g
        ,20.0, 0.0))\n',imageSize(2));
46  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumAxisByTwoPoint(point1=mdb.
        models[''Model-1''].parts[''Part-1''].datums[3], point2=mdb.models[''Model-1''].
        parts[''Part-1''].datums[2])\n');
47  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].DatumPlaneByPrincipalPlane(offset
        =0.0,principalPlane=XYPLANE)\n');
48  fprintf(fid,'mdb.models[''Model-1''].ConstrainedSketch(name=''__profile__'', sheetSize=%
        g)\n',SS);
49  if length(part)>2
50      for j=2:length(part)
51          for i=1:length(part{j})-1
52              fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                    =(%g , %g), point2=(%g , %g)) \n',part{j}(i,1),part{j}(i,2),part{j}(i
                    +1,1),part{j}(i+1,2));
53          end
54      end
55  end
56  if ~isempty(Dis)
57      for j=1:length(Dis)
58          for i=1:length(Dis{j})-1
59              fprintf(fid,'mdb.models[''Model-1''].sketches[''__profile__''].Line(point1
                    =(%g, %g), point2=(%g, %g)) \n',Dis{j}(i,1),Dis{j}(i,2),Dis{j}(i+1,1),
                    Dis{j}(i+1,2));
60          end
61      end
62  end
63  fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].PartitionFaceBySketch(faces=\n');
64  fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt(((%g, %g, 0.0),
```

```
    )),\n',part{1}(1,1),part{1}(1,2)); %?
fprintf(fid,'    sketch=mdb.models[''Model-1''].sketches[''__profile__''], \n');
fprintf(fid,'    sketchUpEdge=mdb.models[''Model-1''].parts[''Part-1''].datums[4])\n');
fprintf(fid,'del mdb.models[''Model-1''].sketches[''__profile__''] \n');
if ~isempty(Dis)
    for i=1:length(Dis)
        in = inpolygon(Discontinuity{i,4}(:,1),imageSize(1)-Discontinuity{i,4}(:,2),
            Discontinuity{i,2}(:,1),Discontinuity{i,2}(:,2));
        id = find(in==1);
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].RemoveFaces(deleteCells=
            False, faceList=(\n');
        fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].faces.findAt((%g, %g,
             0.0), ), ))\n', Discontinuity{i,4}(id(1),1),imageSize(1)-Discontinuity{i
            ,4}(id(1),2));
    end
end
for j=1:size(Domains,1)
    fprintf(fid,'mdb.models[''Model-1''].Material(name=''Material-%g'')\n',j);
    fprintf(fid,'mdb.models[''Model-1''].materials[''Material-%g''].Density(table=((1, )
        , ))\n',j);
    fprintf(fid,'mdb.models[''Model-1''].materials[''Material-%g''].Elastic(table=((1,
        1), ))\n',j);
end
for j=1:size(Domains,1)
    fprintf(fid,'mdb.models[''Model-1''].HomogeneousShellSection(idealization=
        NO_IDEALIZATION, \n');
    fprintf(fid,'    integrationRule=SIMPSON, material=''Material-%g'', name=''Section-%
        g'', \n',j,j);
    fprintf(fid,'    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF,
        temperature=\n');
    fprintf(fid,'    GRADIENT, thickness=1000.0, thicknessField='''', thicknessModulus=
        None, \n');
    fprintf(fid,'    thicknessType=UNIFORM, useDensity=OFF)\n');
end
for j=1:size(Domains,1)
    if j==1
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
            Model-1''].parts[''Part-1''].faces.findAt(((%g , %g , 0.0), )), name=''Set-1
            '') \n',Domains{j,2}(1,1),Domains{j,2}(1,2));
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
            =0.0, \n');
        fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
        fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-1''],
            sectionName=\n');
        fprintf(fid,'    ''Section-1'', thicknessAssignment=FROM_SECTION)\n');
    else
        in = inpolygon(Domains{j,4}(:,1),imageSize(1)-Domains{j,4}(:,2),Domains{j
            ,2}(:,1),Domains{j,2}(:,2));
        id = find(in==1);
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].Set(faces=mdb.models[''
            Model-1''].parts[''Part-1''].faces.findAt(((%g , %g , 0.0), )), name=''Set-%
            g'') \n',Domains{j,4}(id(1),1),imageSize(1)-Domains{j,4}(id(1),2),j);
        fprintf(fid,'mdb.models[''Model-1''].parts[''Part-1''].SectionAssignment(offset
            =0.0, \n');
        fprintf(fid,'    offsetField='''', offsetType=MIDDLE_SURFACE, region=\n');
        fprintf(fid,'    mdb.models[''Model-1''].parts[''Part-1''].sets[''Set-%g''],
            sectionName=\n',j);
        fprintf(fid,'    ''Section-%g'', thicknessAssignment=FROM_SECTION)\n',j);
    end
end
fclose(fid);
end
```

## Code S20: Manual Counting (for Validation of WingSegment)

Code S20 was developed for manually counting cells and junctions on insect wings, which was achieved through mouse clicking to mark cells and junctions. We employed this code to validate the automated cell and junction counting of WingSegment. The input for this

code is the wing image, and the output consists of the coordinates of the marked cells and junctions.

Code S20: Custom code for manual marking of cells and junctions.

```
function Manual_Counter
[file,path,indx] = uigetfile('*.*');
if indx
    w = imread([path file]);
else
    return
end
imshow(w)
prompt = {'What do you want to count?'};
dlgtitle = 'Input';
dims = [1 35];
definput = {''};
countingObject = inputdlg(prompt,dlgtitle,dims,definput);
e=1;
N = 0;
xAll=[]; yAll=[];
while e
    [x,y]=getpts;
    xAll=[xAll ; x];
    yAll=[yAll ; y];
    hold on
    plot(x,y,'ro','markerfacecolor','r')
    N = N+ size(x,1);
    answer = questdlg('Would you like to continue counting?', ...
        'Continue or Stop?',...
        'Yes, please', 'No thank you','Yes, Please');
    switch answer
        case 'Yes, please'
            disp([answer ' coming right up.'])
            e = 1;
        case 'No thank you'
            disp([answer ' coming right up.'])
            e = 0;
    end
end
f = msgbox(['Number of ' countingObject{1} ': ' num2str(N)]);
writematrix([xAll yAll],[countingObject{1} ' Coordinates.csv'])
```

---

## Code S21: Burning Algorithm

Code S21: Custom code for manual marking of cells and junctions.

```
function Run_BA
w=imread('sample.png');
w=rgb2gray(w);
w=double(w);
w=w/255;
id1 = w>0.5;id2 = w<=0.5;
w(id1)=1;
w(id2)=0;
[~,~]=BA(w);
end
function [im,border]=BA(im)
imshow(im)
[x,y]=getpts;
x=round(x); y=round(y);
FWP=[x,y];      % FWP:found white pixel(first white pixel is choosen by user)
im(y,x)=0.2;
border=[];
while ~isempty(FWP)
    X_plus=[FWP(:,1)    ;  FWP(:,1)+1   ;   FWP(:,1)     ; FWP(:,1)-1];
    Y_plus=[FWP(:,2)-1  ;   FWP(:,2)    ;  FWP(:,2)+1   ;   FWP(:,2)];
    coord=[X_plus Y_plus];
```

```
    coord=unique(coord,'rows');
    l=size(coord,1);
    new_FWP=[];
    new_FBP=[];
    for i=1:l
        if im(coord(i,2),coord(i,1))==1
            im(coord(i,2),coord(i,1))=0.2;
            new_FWP=[new_FWP; coord(i,1) coord(i,2)];    %new_FWP: new Found White Pixel
        elseif im(coord(i,2),coord(i,1))==0
            im(coord(i,2),coord(i,1))=0.1;
            new_FBP=[new_FBP; coord(i,1) coord(i,2)];     %new_FBP: new Found Black Pixel
        end
    end
    X_cross=[FWP(:,1)-1; FWP(:,1)-1 ;FWP(:,1)+1 ;FWP(:,1)+1];
    Y_cross=[FWP(:,2)-1 ;FWP(:,2)+1 ;FWP(:,2)-1 ;FWP(:,2)+1];
    coord=[X_plus Y_plus];
    coord=unique(coord,'rows');
    l=size(coord,1);
    for i=1:l
        if im(coord(i,2),coord(i,1))==0
            im(coord(i,2),coord(i,1))=0.1;
            kk(coord(i,2),coord(i,1))=0;
            new_FBP=[new_FBP;coord(i,1) coord(i,2)];     %new_FBP: new Found Black Pixel
        end
    end
    border=[border;new_FBP];
    FWP=new_FWP;
    getframe;
end
end
```

## Code S22: Zhang-Suen Thinning Algorithm

Code S22: Custom code for Zhang-Suen Thinning Algorithm.

```
function W_thined=Zhang_Suen_ThinningMethod
[file,path] = uigetfile('*.*');
w= imread([path file]);
if size(w,3)==3
    w= rgb2gray(w);
end
w= double(w)/255;
id1 = w>0.3  ;
id2 = w<=0.3 ;
s = size(w);
w(id1) = 0;
w(id2) = 1;
wOrg = w;
IdB = w==0;
sIdBlackOld = sum(sum(IdB));
e=2;
i=0;
while e~=0
    w2 = w(1:end-2,2:end-1);
    w3 = w(1:end-2,3:end);
    w4 = w(2:end-1,3:end);
    w5 = w(3:end,3:end);
    w6 = w(3:end,2:end-1);
    w7 = w(3:end,1:end-2);
    w8 = w(2:end-1,1:end-2);
    w9 = w(1:end-2,1:end-2);
    B = w2+w3+w4+w5+w6+w7+w8+w9;
    id1 = B>=2 & B<=6;
    % (b) A( p1) =1
    id21 = (w2-w3) == -1;
    id22 = (w3-w4) == -1;
    id23 = (w4-w5) == -1;
    id24 = (w5-w6) == -1;
    id25 = (w6-w7) == -1;
```

```
35      id26 = (w7-w8) == -1;
36      id27 = (w8-w9) == -1;
37      id28 = (w9-w2) == -1;
38      A = id21+id22+id23+id24+id25+id26+id27+id28;
39      id2 = A==1;
40      if rem(i,2)==0
41          C = w2.*w4.*w6;
42          id3 = C==0;
43          D = w4.*w6.*w8;
44          id4 = D==0;
45      else
46          C = w2.*w4.*w8;
47          id3 = C==0;
48          D = w2.*w6.*w8;
49          id4 = D==0;
50      end
51      idAll = id1.*id2.*id3.*id4;
52      id = idAll==1;
53      id = [zeros(1,s(2)-2); id ; zeros(1,s(2)-2)];
54      id = [zeros(s(1),1) id zeros(s(1),1)];
55      w(logical(id)) = 0;
56      IdB = w==0;
57      sIdBlack = sum(sum(IdB));
58      e = sIdBlack-sIdBlackOld;
59      sIdBlackOld = sIdBlack;
60      i = i+1;
61      imshow(~w)
62      drawnow
63  end
64  W_thined=~w;
65  end
```

# Supplementary Tables

The tables provided in this section present the documented values corresponding to the results and Figures presented in the main manuscript. Specifically, Table S1 contains regression and NRMSE values for area, length, width, and circularity of wing cells in *Apis mellifera* (honeybee, Hymenoptera; forewings), *Ischnura elegans* (blue-tailed damselfly, Odonata: Zygoptera; forewings), *Crocothemis erythraea*(Scarlet dragonfly, Odonata: Anisoptera; forewings), and *Schistocerca gregaria* (desert locust, Orthoptera; hindwings), along with mean distances, standard deviations of mean distances, subtracted values of cell and junction numbers, and differences in wing areas. These results serve as the basis for Figure 3 in the main text. Table S2 encompasses measurements made by WingAnalogy, including wing area, length, width, perimeter, cell and junction counts for the left and right wings of honeybee, damselfly, dragonfly, and locust. Table S3 presents asymmetry values for defined sets of wings, which are subsequently visualized in Figures 4 and 5 in the main text. Tables S4, S5, and S6 contain validation results for WingAnalogy, focusing on wing image segmentation and asymmetry measurement. These validation tests were conducted using honeybee wing images.

Table S1: The values that correspond to those displayed in Figure 6.4

| Asymmetry metrics | *Apis mellifera* | *Ischnura elegans* | *Crocothemis erythraea* | *Schistocerca gregaria* |
|---|---|---|---|---|
| Regression(Area) | 0.9748 | 0.9804 | 0.9372 | 0.7394 |
| Regression(Length) | 0.9967 | 0.9841 | 0.9459 | 0.7453 |
| Regression(Width) | 0.9871 | 0.9748 | 0.8854 | 0.7169 |
| Regression(Circularity) | 0.9867 | 0.9265 | 0.8777 | 0.5309 |
| NRMSE(Area) | 0.2307 | 0.2019 | 0.3545 | 0.7419 |
| NRMSE(Length) | 0.1009 | 0.1820 | 0.3310 | 0.7251 |
| NRMSE(Width) | 0.1637 | 0.2268 | 0.4803 | 0.7752 |
| NRMSE(Circularity) | 0.1684 | 0.3850 | 0.4955 | 1.0051 |
| MD(Cells centroid) | 0.0452 | 0.0739 | 0.1827 | 0.3218 |
| MD(Junctions) | 0.0539 | 0.0763 | 0.1514 | 0.2638 |
| MD(Wing outlines) | 0.1699 | 0.1540 | 0.4043 | 0.4161 |
| MD STD(Cells centroid) | 0.0233 | 0.0548 | 0.1147 | 0.1650 |
| MD STD(Junctions) | 0.0405 | 0.0547 | 0.0912 | 0.1499 |
| MD STD(Wing outlines) | 0.1336 | 0.1708 | 0.3651 | 0.3443 |
| Subtract value (Cells) | 0.0000 | 6.0000 | 10.0000 | 53.0000 |
| Subtract value (Junctions) | 0.0000 | 2.0000 | 7.0000 | 22.0000 |
| Wings area differences | 0.4228 | 0.1507 | 1.1724 | 9.9337 |

Table S2: The outcome of measuring the overall geometric features extracted by WingAnalogy for the wings.

| | *Apis mellifera* | | *Ischnura elegans* | | *Crocothemis erythraea* | | *Schistocerca gregaria* | |
|---|---|---|---|---|---|---|---|---|
| | left | right | left | right | left | right | left | right |
| Wing area ($mm^2$) | 17.4124 | 17.8352 | 33.1788 | 33.0280 | 227.8660 | 229.0385 | 464.8849 | 454.9512 |
| Wing length ($mm$) | 9.1468 | 9.1355 | 13.0947 | 13.1151 | 34.1897 | 34.1963 | 36.8002 | 37.3204 |
| Wing width ($mm$) | 1.9037 | 1.9523 | 2.5338 | 2.5183 | 6.6647 | 6.6977 | 12.6327 | 12.1904 |
| Wing perimeter ($mm$) | 20.2807 | 20.4033 | 28.7270 | 28.8562 | 74.8585 | 74.9240 | 103.2428 | 105.6995 |
| Number of cells | 13 | 13 | 140 | 142 | 447 | 440 | 645 | 667 |
| Number of junctions | 24 | 24 | 212 | 218 | 825 | 815 | 1159 | 1212 |

Table S3: Data of Figure 6.5

| | | *Apis mellifera* | *Ischnura elegans* | *Crocothemis erythraea* | *Schistocerca gregaria* |
|---|---|---|---|---|---|
| Set 1 | Regression(Area) | 0.8291 | 0.9899 | 0.9173 | 0.7659 |
| | Regression(Length) | 0.9945 | 0.9987 | 0.9002 | 0.7683 |
| | Regression(Width) | 0.5212 | 0.9848 | 0.9461 | 0.7315 |
| | Regression(Circularity) | 0.3198 | 0.9915 | 0.9283 | 0.2647 |
| | NRMSE(Area) | 0.8752 | 0.1568 | 0.4020 | 0.7262 |
| | NRMSE(Length) | 0.1127 | 0.0604 | 0.4424 | 0.6976 |
| | NRMSE(Width) | 1.3139 | 0.1817 | 0.3316 | 0.7894 |
| | NRMSE(Circularity) | 1.0096 | 0.1321 | 0.3898 | 1.6260 |
| | MD(Centroid) | 0.0554 | 0.0500 | 0.1677 | 0.3439 |
| | MD STD(Centroid) | 0.0419 | 0.0383 | 0.1540 | 0.1770 |
| Set 2 | Regression(Area) | 0.9996 | 0.9890 | 0.7982 | 0.6977 |
| | Regression(Length) | 0.9984 | 0.9676 | 0.9551 | 0.6748 |
| | Regression(Width) | 0.9987 | 0.9926 | 0.7623 | 0.7518 |
| | Regression(Circularity) | 0.9589 | 0.9740 | 0.9183 | 0.6428 |
| | NRMSE(Area) | 0.2363 | 0.1546 | 0.6473 | 0.8479 |
| | NRMSE(Length) | 0.2626 | 0.2608 | 0.3025 | 0.8583 |
| | NRMSE(Width) | 0.1764 | 0.1278 | 0.6713 | 0.7554 |
| | NRMSE(Circularity) | 0.2997 | 0.2587 | 0.3976 | 0.9174 |
| | MD(Centroid) | 0.0489 | 0.0638 | 0.2111 | 0.3577 |
| | MD STD(Centroid) | 0.0185 | 0.0564 | 0.1421 | 0.1620 |
| Set 3 | Regression(Area) | 1.0000 | 0.9738 | 0.8492 | 0.4716 |
| | Regression(Length) | 0.9980 | 0.8582 | 0.9050 | 0.5093 |
| | Regression(Width) | 0.9987 | 0.9893 | 0.5706 | 0.5575 |
| | Regression(Circularity) | 0.9840 | 0.6899 | 0.8203 | 0.5192 |
| | NRMSE(Area) | 0.0696 | 0.2453 | 0.5617 | 1.1361 |
| | NRMSE(Length) | 0.1343 | 0.5296 | 0.4470 | 0.9803 |
| | NRMSE(Width) | 0.0693 | 0.1627 | 0.9151 | 1.0020 |
| | NRMSE(Circularity) | 0.2269 | 0.7464 | 0.5875 | 0.9560 |
| | MD(Centroid) | 0.0390 | 0.0834 | 0.2100 | 0.3446 |
| | MD STD(Centroid) | 0.0135 | 0.0548 | 0.1203 | 0.1553 |
| Set 4 | Regression(Area) | 1.0000 | 0.9660 | 0.8658 | 0.6779 |
| | Regression(Length) | 1.0000 | 0.8645 | 0.8590 | 0.6290 |
| | Regression(Width) | 1.0000 | 0.9735 | 0.7853 | 0.6324 |
| | Regression(Circularity) | 1.0000 | 0.8754 | 0.7185 | 0.4014 |
| | NRMSE(Area) | 0.3135 | 0.3175 | 0.5447 | 0.8331 |
| | NRMSE(Length) | 0.1159 | 0.5754 | 0.5567 | 0.8497 |
| | NRMSE(Width) | 0.9568 | 0.2570 | 0.6708 | 0.8650 |
| | NRMSE(Circularity) | 0.3080 | 0.6397 | 0.7501 | 1.0744 |
| | MD(Centroid) | 0.0433 | 0.0615 | 0.1902 | 0.3005 |
| | MD STD(Centroid) | 0.0355 | 0.0467 | 0.1038 | 0.1415 |
| Set 5 | Regression(Area) | 1.0000 | 0.9561 | 0.9438 | 0.5952 |
| | Regression(Length) | 1.0000 | 0.8927 | 0.9324 | 0.6642 |
| | Regression(Width) | 1.0000 | 0.9276 | 0.8367 | 0.6283 |
| | Regression(Circularity)- | 1.0000 | 0.4067 | 0.7519 | 0.5356 |
| | NRMSE(Area) | 0.2345 | 0.3244 | 0.3396 | 0.8209 |
| | NRMSE(Length) | 0.2597 | 0.5042 | 0.3687 | 0.7880 |
| | NRMSE(Width) | 0.2176 | 0.3943 | 0.5740 | 0.8141 |
| | NRMSE(Circularity) | 0.5245 | 1.1025 | 0.6909 | 0.9612 |
| | MD(Centroid) | 0.0352 | 0.0934 | 0.1538 | 0.2154 |
| | MD STD(Centroid) | 0.0021 | 0.0511 | 0.0801 | 0.1140 |

Table S4: This table compares measurements obtained for the right wing in Figure S27, including area, length, and circularity, using both WingAnalogy and ImageJ.

| | Measured by WingAnalogy | | | Measured by ImageJ | | | Accuracy % | | |
|---|---|---|---|---|---|---|---|---|---|
| Cells labels | Area | Length | Circularity | Area | Length | Circularity | Area | Length | Circularity |
| 1 | 6345 | 268.27 | 0.23 | 6348 | 266.71 | 0.24 | 0.05 | 0.58 | 4.17 |
| 2 | 1547 | 81.74 | 0.53 | 1547 | 79.46 | 0.57 | 0.00 | 2.87 | 7.02 |
| 3 | 5540 | 235.6 | 0.26 | 5543 | 234.03 | 0.27 | 0.05 | 0.67 | 3.70 |
| 4 | 3561 | 141.17 | 0.42 | 3563 | 140.01 | 0.44 | 0.06 | 0.83 | 4.55 |
| 5 | 2097 | 106.25 | 0.43 | 2100 | 103.92 | 0.45 | 0.14 | 2.24 | 4.44 |
| 6 | 2115 | 111.61 | 0.42 | 2116 | 108.63 | 0.45 | 0.05 | 2.74 | 6.67 |
| 7 | 11343 | 199.46 | 0.55 | 11347 | 199.74 | 0.57 | 0.04 | 0.14 | 3.51 |
| 8 | 2876 | 178.84 | 0.21 | 2878 | 174.44 | 0.22 | 0.07 | 2.52 | 4.55 |
| 9 | 5757 | 141.03 | 0.53 | 5760 | 137.06 | 0.55 | 0.05 | 2.90 | 3.64 |
| 10 | 7558 | 160.38 | 0.6 | 7564 | 159.7 | 0.62 | 0.08 | 0.43 | 3.23 |
| 11 | 8302 | 157.28 | 0.65 | 8303 | 154.11 | 0.67 | 0.01 | 2.06 | 2.99 |
| 12 | 5071 | 304.36 | 0.14 | 5076 | 301.94 | 0.15 | 0.10 | 0.80 | 6.67 |
| 13 | 9659 | 196.87 | 0.56 | 9662 | 193.3 | 0.58 | 0.03 | 1.85 | 3.45 |
| | | | | | | **Average** | 0.056 | 1.59 | 3.7 |

Table S5: This table compares measurements obtained for the left wing in Figure S27, including area, length, and circularity, using both WingAnalogy and ImageJ.

| | Measured by WingAnalogy | | | Measured by ImageJ | | | Accuracy % | | |
|---|---|---|---|---|---|---|---|---|---|
| Cells labels | Area | Length | Circularity | Area | Length | Circularity | Area | Length | Circularity |
| 1 | 5906 | 257.381 | 0.237 | 5917 | 254.386 | 0.243 | 0.19 | 1.18 | 2.47 |
| 2 | 1466 | 76.485 | 0.547 | 1467 | 74.888 | 0.591 | 0.07 | 2.13 | 7.45 |
| 3 | 4781 | 227.721 | 0.241 | 4787 | 225.354 | 0.248 | 0.13 | 1.05 | 2.82 |
| 4 | 3301 | 130.465 | 0.448 | 3308 | 129.062 | 0.463 | 0.21 | 1.09 | 3.24 |
| 5 | 1957 | 104.676 | 0.408 | 1958 | 101.056 | 0.432 | 0.05 | 3.58 | 5.56 |
| 6 | 2127 | 116.017 | 0.39 | 2127 | 113.407 | 0.414 | 0.00 | 2.30 | 5.80 |
| 7 | 11424 | 199.812 | 0.51 | 11427 | 198.001 | 0.527 | 0.03 | 0.91 | 3.23 |
| 8 | 2191 | 180.712 | 0.167 | 2197 | 178.987 | 0.172 | 0.27 | 0.96 | 2.91 |
| 9 | 5100 | 134.648 | 0.503 | 5105 | 130.638 | 0.521 | 0.10 | 3.07 | 3.45 |
| 10 | 7334 | 159.38 | 0.581 | 7340 | 157.156 | 0.606 | 0.08 | 1.42 | 4.13 |
| 11 | 7806 | 149.777 | 0.662 | 7808 | 147.363 | 0.689 | 0.03 | 1.64 | 3.92 |
| 12 | 7104 | 306.232 | 0.197 | 7106 | 304.002 | 0.201 | 0.03 | 0.73 | 1.99 |
| 13 | 9386 | 188.82 | 0.573 | 9393 | 187.211 | 0.589 | 0.07 | 0.86 | 2.72 |
| | | | | | | **Average** | 0.096 | 1.6 | 3.8 |

Table S6: This table compares measurements obtained for the left wing in Figure S27, including area, length, and circularity, using both WingAnalogy and ImageJ.

| | | WingAnalogy | ImageJ | Accuracy % |
|---|---|---|---|---|
| Regression | Area | 0.9743 | 0.9744 | 0.01 |
| | Length | 0.9969 | 0.9965 | 0.04 |
| | Circulratiy | 0.9838 | 0.9848 | 0.1 |
| NRMSE | Area | 0.2305 | 0.2298 | 0.3 |
| | Length | 0.1014 | 0.1054 | 3.8 |
| | Circulratiy | 0.1851 | 0.1787 | 3.6 |

# Supplementary Files (Zenodo Repository, Youtube)

## Supplementary Videos

All following videos are available via the following YouTube playlist: https://www.youtube.com/playlist?list=PLWsEPgdSKTKTl8Gag1X8NdlKHiyJBaQ__

- **WingMesh Video S1.** How the Burning Algorithm (BA) works. https://youtu.be/tMcBLBBr-Mw
- **WingMesh Video S2.** A tutorial about the use of WingMesh. https://youtu.be/1peMAMImuyo
- **WingGram Video S1.** A tutorial about the use of WingGram. https://youtu.be/CMiDwAz-MWY
- **WingSegment Video S1.** Manual counting of Scarlet Dragonfly forewing cells. https://youtu.be/386ZQeDOY54
- **WingSegment Video S2.** Manual counting of Scarlet Dragonfly forewing junctions. https://youtu.be/UkcewA7JYM4
- **WingSegment Video S3.** Region Growing visualization. https://youtu.be/UkYFeLoqp48
- **WingSegment Video S4.** Parallel Vein Detection (PVD) visualization. https://youtu.be/MzwJbBsNISw
- **WingSegment Video S5.** Sequential Vein Detection (SVD) visualization. https://youtu.be/a9588hbSBVk
- **WingSegment Video S6.** A quick demo of WingSegment. https://youtu.be/oYfJsns3RMk
- **WingSegment Video S7.** From WingSegment to FreeCAD. This video introduces how to import the generated model. https://youtu.be/B-y7QU5FD6o
- **WingAnalogy Video S1.** Preparing image for WingAnalogy. https://youtu.be/uSgO9pEbk3w
- **WingAnalogy Video S2.** How to define a new project in WinbgAnalogy. https://youtu.be/02UfL1I1Vr4
- **WingAnalogy Video S3.** How to add a new pair in WingAnalogy. https://youtu.be/zpPRYpH-82c
- **WingAnalogy Video S4.** How to use scaling tool in WingAnalogy. https://youtu.be/FqIqJhZT-eg
- **WingAnalogy Video S5.** How to use superimposing tool in WingAnalogy. https://youtu.be/lEBp3zARHts
- **WingAnalogy Video S6.** Statistics and regressions in WingAnalogy. https://youtu.be/PbPZ2cudXXM
- **WingAnalogy Video S7.** How to define cell set in WingAnalogy. https://youtu.be/4ZRlckITJ4A

- **WingAnalogy Video S8.** How to export the result of pair in WingAnalogy. https://youtu.be/S7SY-0uq6gE
- **WingAnalogy Video S9.** How to save a project in WingAnalogy. https://youtu.be/8iy0Ox579i0
- **WingAnalogy Video S10.** How to import a saved project to WingAnalogy. https://youtu.be/LQO8NaG_fe4
- **WingAnalogy Video S11.** How to export the result of all pairs in WingAnalogy. https://youtu.be/M1eHYOY2aVg
- **WingAnalogy Video S12.** How to export the result of all sets in WingAnalogy. https://youtu.be/mXgPLTpRBqE

## Supplementary 3D Models

The following are the supplementary files of five papers described in this thesis documented at online repositories.

- **WingMesh-Model S1.** The *.inp file of Example 1 in Section 3.4.
- **WingMesh-Model S2.** The *.inp file of Example 2 in Section 3.4.
- **WingMesh-Model S3.** The *.inp file of example 3 in Section 3.4.
- **WingMesh-Model S4.** The *.inp file of example 4 in Section 3.4.
- **WingMesh-Model S5.** The *.inp file of example 5 in Section 3.4.
- **WingMesh-Model S6.** The *.inp file of example 6 in Section 3.4.
- **WingMesh-Model S7.** The *.inp file of example 7 in Section 3.4.
- **WingMesh-Model S8.** The *.inp file of example 8 in Section 3.4.
- **WingMesh-Model S9.** The *.inp file of example 9 in Section 3.4.
- **WingMesh-Model S10.** The *.inp file of example 10 in Section 3.4.
- **WingGram-Model S1.** the JNL model of Ephemeroptera in Figure 4.4.
- **WingGram-Model S2.** the JNL model of Dermaptera in Figure 4.4.
- **WingGram-Model S3.** the JNL model of Odonata in Figure 4.4.
- **WingGram-Model S4.** the JNL model of Orthoptera in Figure 4.4.
- **WingGram-Model S5.** the JNL model of Blattodea in Figure 4.4.
- **WingGram-Model S6.** the JNL model of Mecoptera in Figure 4.4.
- **WingGram-Model S7.** the JNL model of Diptera in Figure 4.4.
- **WingGram-Model S8.** the JNL model of Lepidoptera in Figure 4.4.
- **WingGram-Model S9.** the JNL model of Trichoptera in Figure 4.4.
- **WingGram-Model S10.** the JNL model of Coleoptera in Figure 4.4.

- **WingGram-Model S11.** the multi-section JNL model of the damaged wing in Figure 4.5a.
- **WingGram-Model S12.** the two-section JNL model of the damaged wing in Figure 4.5a.
- **WingGram-Model S13.** the JNL model of the fossile wing in Figure 4.5b.
- **WingGram-Model S14.** the corrugated JNL model of the damaged wing in Figure 4.2k.
- **WingGram-Model S15.** the corrugated JNL model of the *(Sympetrum vulgatum* wing in Figure 4.5c.
- **WingGram-Model S16.** the G-Code of the model in Figure 4.5d.
- **WingGram-Model S17.** The CAE file of the model used to validate WingGram.
- **WingSegment-Model S1.** The 3D printable model of Honeybee wing in Figure 5.2g.
- **WingSegment-Model S2.** The 3D printable model of Scarlet Dragonfly hindwing in Figure 5.2e.
- **WingSegment-Model S3.** The 3D printable model of Scarlet Dragonfly forewing in Figure S9.
- **WingSegment-Model S4.** The 3D printable model of a Damselfly wing in Figure S9.
- **WingSegment-Model S5.** The 3D printable model of a Locust wing in Figure S9.

## Supplementary Sample Images

- **WingAnalogy-Sample Image S1.** *Apis mellifera* (honeybee, Hymenoptera; left forewing).
- **WingAnalogy-Sample Image S2.** *Apis mellifera* (honeybee, Hymenoptera; right forewing).
- **WingAnalogy-Sample Image S3.** *Ischnura elegans* (blue-tailed damselfly, Odonata: Zygoptera; left forewing).
- **WingAnalogy-Sample Image S4.** *Ischnura elegans* (blue-tailed damselfly, Odonata: Zygoptera; right forewing).
- **WingAnalogy-Sample Image S5.** *Crocothemis erythraea*(Scarlet dragonfly, Odonata: Anisoptera; left forewing)
- **WingAnalogy-Sample Image S6.** *Crocothemis erythraea*(Scarlet dragonfly, Odonata: Anisoptera; right forewing)
- **WingAnalogy-Sample Image S7.** *Schistocerca gregaria* (desert locust, Orthoptera; left hindwing)

- **WingAnalogy-Sample Image S8.** *Schistocerca gregaria* (desert locust, Orthoptera; right hindwing)
- **WingSegment-Sample Image S1.** Image of a Honeybee wing.
- **WingSegment-Sample Image S2.** Image of a Scarlet Dragonfly wing hindwing.
- **WingSegment-Sample Image S3.** Image of a Scarlet Dragonfly forewing.
- **WingSegment-Sample Image S4.** Image of a Damselfly wing.
- **WingSegment-Sample Image S5.** Image of a Locust wing.

## Other Supplementary Files

- Generated reports by WingAnalogy
  - *WingAnalogy-Honeybee_GeneratedReport.pdf*
  - *WingAnalogy-Damselfly_GeneratedReport.pdf*
  - *WingAnalogy-Dragonfly_GeneratedReport.pdf*
  - *WingAnalogy-Desert locust_GeneratedReport.pdf*
- Comparison raw Data Generated by WingAnalogy
  - *WingAnalogy-Honeybee_ComparisonRawData.txt*
  - *WingAnalogy-Damselfly_ComparisonRawData.txt*
  - *WingAnalogy-Dragonfly_ComparisonRawData.txt*
  - *WingAnalogy-Desert locust_ComparisonRawData.txt*
- Comparison tables Generated by WingAnalogy
  - *WingAnalogy-Honeybee_ComparisonTable.csv*
  - *WingAnalogy-Damselfly_ComparisonTable.csv*
  - *WingAnalogy-Dragonfly_ComparisonTable.csv*
  - *WingAnalogy-Desert locust_ComparisonTable.csv*
- All pairs' results Generated by WingAnalogy
  - *Wing Analogy, all pairs' result.pdf*
- All pairs' sets' results Generated by WingAnalogy
  - *Wing Analogy, all pairs' sets' Result.pdf*
- **WingSegment-File S1.** The coordinate of Scarlet Dragonfly forewing cells centroids, extracted manually using Code S1.
- **WingSegment-File S2.** The coordinate of Scarlet Dragonfly forewing junctions, extracted manually using Code S1.
- **WingSegment-File S3.** The coordinate of Scarlet Dragonfly forewing cells centroids, extracted automatically using WingSegment.

- **WingSegment-File S4.** The coordinate of Scarlet Dragonfly forewing junctions, extracted automatically using WingSegment.
- **WingSegment-File S5.** This file contains all information related to the validation of WingSegment.

I hereby declare that, aside from the guidance provided by my supervisor, the entire dissertation is my original work. The thesis, in part or whole, has not been submitted for examination as a component of any other doctoral degree. The crucial aspects of the work have been published in scientific journals. The thesis adheres to the Rules of Good Scientific Practice of the German Research Foundation. I also declare that no academic degree has ever been withdrawn from me.

Kiel, _____

Shahab Eshghi Sahraei