ISSN 1688-2784

UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Application of Image Processing and Artificial Intelligence Techniques for the Automatic Dendrometry of Native and Commercial Wood Species

TESIS PRESENTADA A LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE LA REPÚBLICA POR

Henry Marichal

EN CUMPLIMIENTO PARCIAL DE LOS REQUERIMIENTOS PARA LA FINALIZACIÓN DE LA CARRERA DE DOCTORADO EN INGENIERÍA ELÉCTRICA.

DIRECCIÓN DE TESIS

Gregory Randall . . . . . Universidad de la República
Diego Passarella . . . . . Universidad de la República

TRIBUNAL

Álvaro Gómez . . . . . Universidad de la República
Alexander Gillert (Revisor Externo) University of California, San Diego, USA
Graciela Muñiz (Revisor Externo) . . . . . Universidade Federal do Paraná, Brasil

DIRECCIÓN ACADÉMICA

Gregory Randall . . . . . Universidad de la República

Montevideo
Wednesday 10th September, 2025

Application of Image Processing and Artificial Intelligence Techniques for the Automatic Dendrometry of Native and Commercial Wood Species, Henry Marichal.

ISSN 1688-2784

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.2).
Contiene un total de 144 páginas.
Compilada el Wednesday 10th September, 2025.
http://iie.fing.edu.uy/

Life is about momentum. Pick a direction and don´t let anything -man or storm- turn you aside.

DALINAR KHOLIN

Esta página ha sido intencionalmente dejada en blanco.

Acknowledgments

En estas líneas quiero expresar mi sincero agradecimiento a todas las personas que me apoyaron a lo largo de esta etapa.

En primer lugar, a mi familia y amigos, quienes fueron un apoyo y una compañía constante en los momentos de dificultad. En particular, a Juan Cosentino, Andrés Bologna, Fabián Vique, Andreina Bazzino, Facundo Marichal, Gastón Rial, Matías Pérez, Enzo Bruno, Tania Aguirre, Nicolás Camejo, Gabriela Cabeda y Martín De León por su apoyo incondicional. A mis padres, Rosario Camejo y Henrry Marichal, por acompañarme y por intentar comprender la importancia de la formación continua. A mi psicóloga, Agustina Carracedo, por ayudarme a entender que somos seres emocionales que razonan.

A mi tutor, Gregory Randall, quien durante el doctorado me guió y estuvo siempre presente cuando lo necesité. Al resto de los integrantes del grupo UruDendro, que asumieron la ardua tarea de adquisición y anotación de imágenes: Diego Passarella, Christine Lucas, Ludmila Profumo, Verónica Casaravilla, María Noel Rocha Galli, Karolain Mello, Joaquín Blanco y Serrana Ambite. Agradezco también a quienes compartieron ideas y discusiones que enriquecieron esta tesis y me ayudaron a mejorar mi trabajo: Guillermo Sapiro, Matías Di Martino, Ariel Stassi, Mateo Nogueira, Alfredo Solari, Candice Power, Ignacio Ramírez, José Luis Nunes y Francesco Franzoni.

Quiero agradecer a la ANII por el financiamiento que hizo posible culminar mis estudios de doctorado, al fondo IFUMI que apoyó mi estadía en París y a la Universidad de Duke, que me brindó la oportunidad de viajar a Durham e intercambiar con sus estudiantes. También agradezco a las empresas en las que trabajé en paralelo durante estos años, las cuales me ofrecieron la estabilidad económica y laboral necesaria para poder enfocarme en mis estudios: Digital Sense e Isbel S.A.

Finalmente, expreso mi profundo agradecimiento al Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República, donde recibí toda mi formación universitaria.

Esta página ha sido intencionalmente dejada en blanco.

A mi familia.

Esta página ha sido intencionalmente dejada en blanco.

Summary

Tree-ring analysis is a cornerstone of dendrometry, providing essential information for dendrochronology, forest dynamics, and growth studies. Traditionally, ring marking is performed manually, a process that is time-consuming, subjective, and difficult to scale to large image datasets. Moreover, annual growth measurements are often taken in a one-dimensional manner, which complicates comparisons among samples.

This thesis proposes the use of ring area as an alternative to ring width, since the latter is a one-dimensional measure that is difficult to standardize. Image-processing algorithms were developed to accurately delineate annual growth curves and calculate growth area in both trees and shrubs. A graphical interface was implemented to combine automatic detection with manual correction tools when needed. Annotated image databases for several species were also created, enabling systematic evaluation of the algorithms. Finally, a case study in ecology–climatology is presented, comparing ring width and ring area as climate indicators.

Esta página ha sido intencionalmente dejada en blanco.

Preface

This work focuses on the delineation of growth rings in images of tree cross-sections (discs). While the literature includes various studies on ring delineation in cores, this approach, commonly used in dendrochronology due to its non-destructive nature, is not feasible for all species. In particular, in shrubs, it is not possible to extract cores, and the only viable way to analyze growth rings is through full cross-sections.

Furthermore, in forestry research, where the primary goal is to evaluate different silvicultural techniques to increase timber production, the analysis of ring area can provide more informative insights than just ring width. This is particularly true in cases where the pith is highly eccentric (i.e., the geometric center of the disc does not coincide with the pith). At the time this project began, growth studies in the Forest engineering degree, of the Northeast Regional University Centre, were carried out manually by measuring ring widths along four to eight radial directions on physical discs. This motivated the need for a tool capable of extracting such measurements automatically.

In February 2021, when this project started, there were no available automatic methods for delineating rings in full cross-section images. The only public dataset available at that time consisted of 7 annotated images of *Abies alba* released by Kennel et al. [26], similar in nature to a small dataset of 64 *Pinus taeda L.* images we had access to.

Thus, the first two main tasks of this thesis were to support the creation of a new annotated image dataset and, in parallel, to develop an automatic ring delineation algorithm. These tasks, including the initial documentation, were completed by May 2023. The method was named CS-TRD, and the dataset was named UruDendro (UruDendro1 hereon).

One month later, in June 2023, Gillert et al. [17] introduced the INBD method, based on convolutional neural networks, which performs automatic delineation of rings in cross-sections of shrubs, rings that visually resemble those found in conifers such as *Pinus taeda* and Abies alba. We therefore studied and trained this method on the UruDendro dataset to compare it against CS-TRD. INBD achieved excellent results, with an F-score of 94.5% when the pith location (associated with the innermost ring) was manually provided. In comparison, CS-TRD, based on classical image processing techniques, obtained an F-score of 93.1% under the same conditions.

The fourth task involved evaluating existing automatic pith detection methods and developing a new one. Among open-source implementations, we tested the method by Decelle et al. [6], designed for field or sawmill settings. We also implemented the algorithm by Schraml and Uhl [54] and developed a new method named APD [41]. Additionally, we trained a YOLOv8 model inspired by Decelle's approach for pith detection.

As the fifth task, we implemented a graphical interface tool that, given an image of

a cross-section, computes forestry-relevant indicators based on ring delineation, such as area, perimeter, and width. The tool also includes an interface for manually correcting or adding rings that were inaccurately detected by the automatic methods.

We also extended the CS-TRD method by replacing its edge segmentation step with one based on the U-Net convolutional neural network architecture [53]. This new variant, named DeepCS-TRD, achieved an F-score of 96.1% on the UruDendro1 dataset.

Finally, we contributed to the creation of annotated image datasets for three species: *Gleditsia triacanthos L.*, *Salix glauca L.*, and an extended set of *Pinus taeda* images.

Contents

Esta página ha sido intencionalmente dejada en blanco.

Chapter 1

Introduction

1.1 Terminology

This work focuses on developing tools for dendrometry, specifically the analysis of growth rings in trees and shrubs, using images of cross-sections (discs). In this section, we define the terminology used throughout the manuscript. Figure 1.1 shows a cross-section image of a *Pinus taeda L.* species. A rectangular region outlined in red highlights what is commonly referred to as a core sample. In the literature, tree growth is often analyzed using core samples because this method is non-destructive, meaning it does not require felling the tree.

A variety of structures can be observed in a disc. One is the pith, shown in Figure 1.2b highlighted in red. This structure corresponds to the first year of the tree's growth. Additionally, the disc contains annual growth rings, also known as growth rings, which are marked in black in Figure 1.2b. In some species, such as *P. taeda*, annual growth is composed of two distinct zones, depending on the time of year when growth occurred: earlywood, formed during spring and summer and characterized by a lighter color; and latewood, formed during autumn and winter, characterized by a darker color (see Figure 1.2c and Figure 1.3 for more details). Another structure visualized in RGB images is reaction wood, which exhibits distinct physical, chemical, and mechanical properties compared to the rest of the wood. It is highlighted in green in Figure 1.3c.

Other features may be present in a disc, such as knots (Figure 1.4a), cracks (Figure 1.4b), or fungal stains (Figure 1.4c). While each species has specific characteristics, these are among the most common terms in this field. All the above definitions can be found in the glossary (B.2.5).

1.2 Objective

To develop an open-source tool capable of performing automatic or semi-automatic measurements on growth rings in cross-section images, including area, perimeter, and ring width.

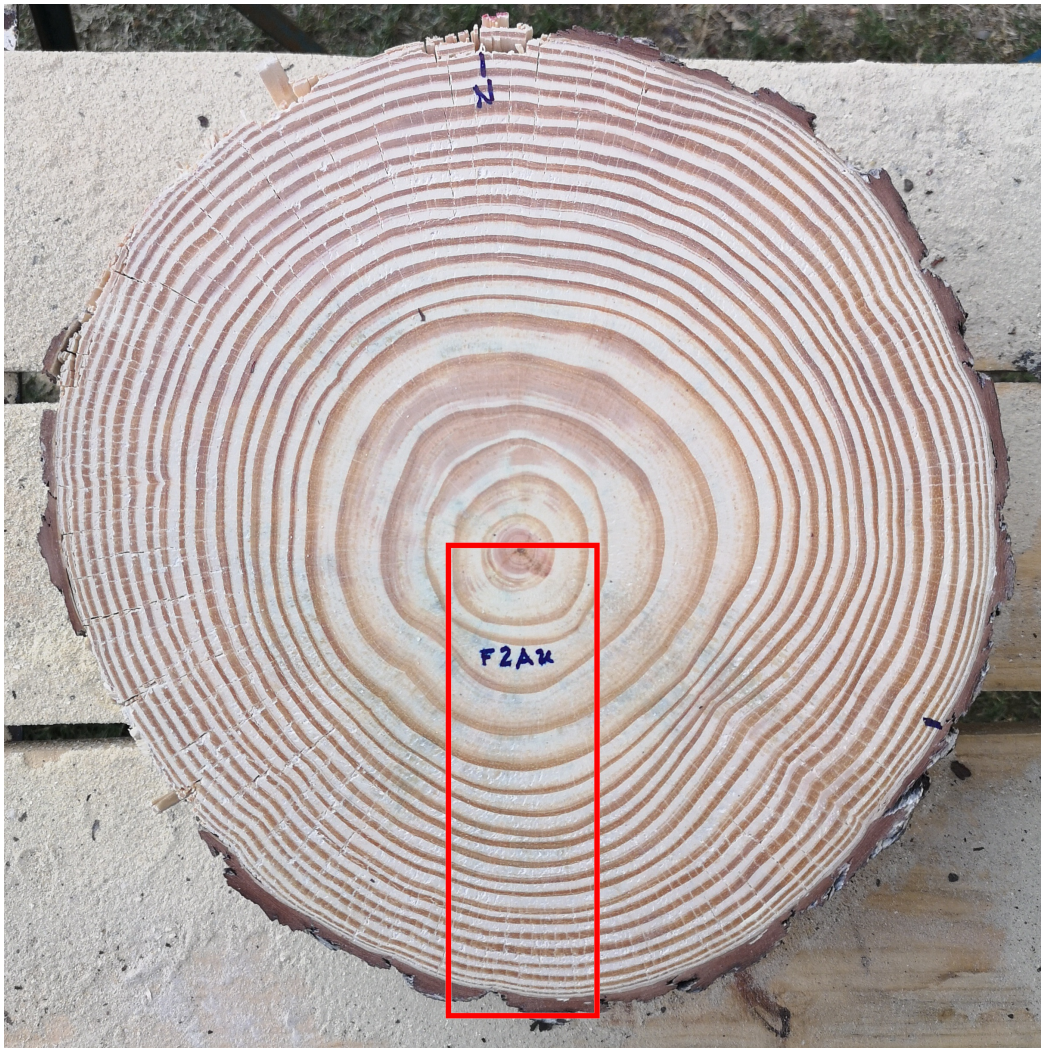
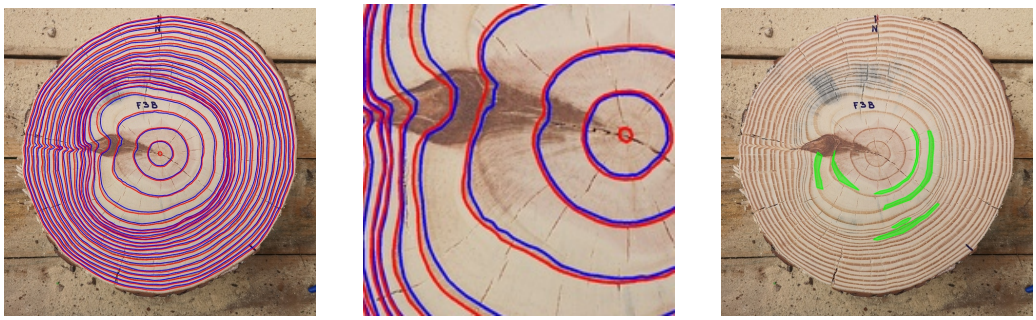


Figure 1.1: Image of a tree disc or cross-section. The red box highlights a portion known as a core sample.

(a) (b) (c)

Figure 1.2: (a) Original disc image. (b) Annual growth rings in black, with the pith outlined in red. (c) The annual ring comprises two regions: earlywood (green) and latewood (blue).



(a) (b) (c)

Figure 1.3: Image cross-section annotations. (a) Annual rings are highlighted in red, and the Earlywood-Latewood boundary is highlighted in blue. (b) Zoom In. (c) Reaction wood is highlighted in green.

(a) (b) (c)

Figure 1.4: (a) Disc sample with a knot and fungus presence. (b) Disc sample with a crack. (c) Disc sample with fungus presence

1.3 Approach

Figure 1.5 presents a general overview of the tasks addressed in this work. At the beginning of this project, no public implementation was available for detecting growth rings in cross-sectional images. Furthermore, the only available image dataset consisted of just seven cross-section images without annotated annual rings [26][1]. Access to annotated datasets is essential for training and evaluation in the development of automatic methods. Therefore, one of the main tasks of this project was the creation of image datasets with their corresponding annotations (see Figure 1.5a), which is detailed in Chapter 2. The second task involved the development of algorithms for detecting growth rings, including the pith detection as a subtask (see Figure 1.5b and c). Automatic pith detection is addressed in Chapter 3, while ring detection methods are described in Chapter 4. Finally, this work aimed to develop a graphical user interface (GUI) enabling non-programmers to interactively load a cross-section image, execute automatic ring detection algorithms, edit/remove/add ring boundaries, and compute relevant metrics such as area, perimeter, or ring width. The GUI is detailed in Chapter 5.

[1] Annotated ring curves were available, but could not be successfully loaded.

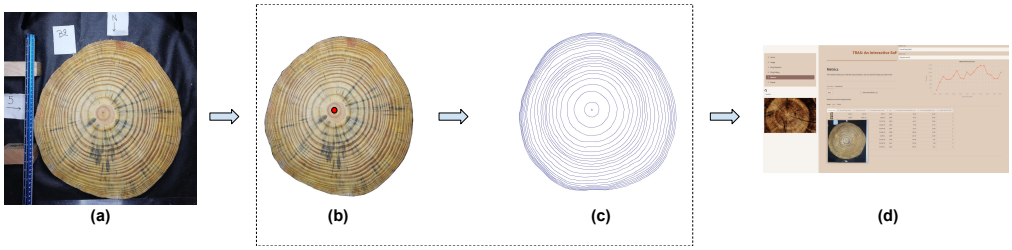


Figure 1.5: (a) Generation of image datasets with annotations. (b) and (c) Development of algorithms for growth ring (blue curves) and pith (red dot) detection. (d) Development of a GUI integrating automatic detection and interactive corrections, along with measurement computation.

Each chapter mentioned above is self-contained and includes a contextual introduction, the methodology employed, the results obtained, and corresponding conclusions.

1.4 Contributions

This section summarizes the main contributions of each task described above. For more detailed information, refer to the corresponding chapter.

Regarding image dataset generation and annotation (see Chapter 2), the contributions are:

- 117 annotated images of *P. taeda* including annual rings, earlywood-latewood boundary and reaction wood.
- 102 annotated images of *P. taeda* with annual ring boundaries.
- 17 annotated images of *Gleditsia triacanthos L.*, with over 70 additional images acquired but pending annotation.
- 50 annotated images of *Salix glauca L.*.

The author contributed to the annotation process by participating in the whole process of building the datasets. Assisting in selecting and adapting annotation tools and verifying the annotations in collaboration with species experts. Finally, in organizing the collections as accessible datasets, available for image processing and machine learning.

Regarding automatic pith detection (see Chapter 3), the contributions are:

- Implementation of the method proposed in [54].
- Design and implementation of a novel pith detection algorithm.
- Training of a YOLOv8 deep learning model for pith detection.
- Evaluation and comparison of the methods mentioned above.

Regarding automatic ring detection (see Chapter 4), the contributions are:

- Design and implementation of a classical image processing algorithm named CS-TRD.
- Modification of the above method to use a U-Net architecture, resulting in DeepCS-TRD.
- Training of the INBD deep learning model [17], originally developed for microscopic shrub images.
- Evaluation and comparison of all proposed methods.

Regarding the user-oriented tool (see Chapter 5), the contribution is:

- Implement a graphical user interface with high precision to compute ring metrics such as area, perimeter, and width.

1.5 Publications

This section lists peer-reviewed publications related to this work, including those published, accepted, under review, or submitted.

Published

- **Henry Marichal**, Diego Passarella, Gregory Randall. Automatic Wood Pith Detector: Local Orientation Estimation and Robust Accumulation. *International Conference on Pattern Recognition (ICPR)*, 2024. GitHub.

Accepted

- **Henry Marichal**, Verónica Casaravilla, Candice Power, Karolain Mello, Ludmila Profumo, Joaquín Mazarino, Christine Lucas, Ludmila Profumo, Diego Passarella, Gregory Randall. DeepCS-TRD, a Deep Learning-based Cross-Section Tree Ring Detector. *International Conference on Image Analysis and Processing (ICIAP)*, 2025. GitHub.
- **Henry Marichal**, Diego Passarella, Christine Lucas, Ludmila Profumo, Verónica Casaravilla, María Noel Rocha Galli, Serrana Ambite, Gregory Randall. UruDendro, a public dataset of 64 cross-section images and manual annual ring delineations of Pinus taeda L.. *Annals of Forest Science Journal*, 2025. GitHub.
- **Henry Marichal**, Diego Passarella, Gregory Randall. CS-TRD: a Cross-Sections Tree Ring Detection Method. *Image Processing On Line Journal*, 2025. GitHub.

Submitted

- **Henry Marichal**, Diego Passarella, Gregory Randall. Implementation and discussion of the "Pith Estimation on Rough Log End Images using Local Fourier Spectrum Analysis" method. *Image Processing On Line Journal*, 2024. GitHub.

- **Henry Marichal**, Candice Power, Urs A. Treier, Giulia Resente, Signe Normand and Gregory Randall. Assessing automatic ring detection on microscopy images of *Salix glauca*. *Dendrochronologia Journal*, 2025.
- **Henry Marichal**, Diego Passarella, Gregory Randall. TRAS: An Interactive Software for Tracing Tree Ring Cross Sections. *Dendrochronologia Journal*, 2025. GitHub.

1.6 Organization

This thesis is structured as follows: Chapter 2 presents the image datasets and annotations generated during this study, Chapter 3 discusses the evaluation of automatic pith detection algorithms, and Chapter 4 details the development of automatic ring delineation methods, including CS-TRD, DeepCS-TRD, and INBD. Chapter 5 introduces the tool for computing forestry industry-relevant indicators. Finally, Chapter 6 presents the conclusions of this work. Additional details on pith and ring detection methods are included in Appendices A and B.

Chapter 2

Dataset

This chapter is partially based on the work submitted to the Dendrochronology Journal special issue Trace 2024 entitled “Assessing automatic ring detection on microscopy images of *Salix glauca*” and on the accepted work in Annals of Forest Science Journal “UruDendro, a public dataset of 64 cross-section images and manual annual ring delineations of *Pinus taeda L.*”.

2.1 Introduction

Annotated datasets are mandatory for building tree-ring detection algorithms, are crucial for training and evaluating their performance, and are essential for obtaining accurate results. The datasets are specific for each species under consideration.

All the datasets developed during this thesis were a collective product of the project "FMV_1_2023_1_176061, UruDendro 2.0: Aplicación de técnicas de procesamiento de imágenes e inteligencia artificial para la dendrometría automática de especies de madera nativas y comerciales", funded by the National Agency of Research and Innovation of Uruguay (ANII in Spanish acronym). Besides the author of this Thesis and Prof. Gregory Randall, who are engineers, the other members of the team are forestry and dendrometry specialists.

The construction of an annotated dataset is a complex process that includes:

1. Define the desirable characteristics of the dataset: collect sites, define the number of samples and their characteristics, select and train annotators, define protocols for the annotations, etc.

2. Collect the samples. This step implies field work.

3. Process the samples: cleaning, polishing, and sanding, eventually chemical treatment.

4. Image acquisition. This step means the definition of a procedure involving how to acquire (for example defining an illumination system), select a digital camera or a

scanner, the acquisition process itself, and the implementation of protocols to guarantee that the names of the images, the metadata, and all the sample information are coherently registered.

5. Annotate the acquired images. Sometimes, it also implies the physical annotation of the samples. Eventually, the annotations will be revised by a second expert.

6. Organize the images and annotations so that it is accessible by the users, e.g., in a database.

7. Build a website to publish the dataset so it can be accessed by other researchers.

The first step was discussed collectively within the project team. The author of this thesis led the general process and played a crucial role in steps 6 and 7. He was also responsible for building the tools to facilitate the annotation step and advised the rest of the team for the fourth step. The (very time-consuming) steps 2, 3, and 4 were carried out by the team members with specific knowledge to do so. The whole process took several months for each collection.

This chapter describes the image datasets developed during the course of this thesis. When we began this work in February 2021, we observed a significant scarcity of annotated images for tree cross-sections and cores. As this thesis focuses on developing algorithms for complete cross-sections, we need annotated datasets of this type of sample.

At the start of this work, the only available dataset was the one proposed by Kennel et al. [26], which includes seven images of the species *Abies alba*, its annual ring annotations, and a delineation method. However, the code was unavailable. We were unable to process the authors' annotations, so we generated our own (see the last row in Table 2.1).

In June 2023, Gillert et al, [17] released a dataset of 213 high-resolution annual ring annotated images of cross-sections from three shrub species (*Dryas octopetala*, *Empetrum hermaphroditum*, and *Vaccinium myrtillus*) along with an algorithm for delineating tree rings in this dataset.

Longuetaud et al. [35] released the $TreeTrace\_Douglas$ database in November 2022, including several wood cross-section image datasets. This dataset includes images and measurements, such as growth ring width and pith pixel location, along with other wood properties, which were acquired at various stages of Douglas-fir log processing in a sawmill. However, it does not provide delineations of tree rings in the images. A few months later, in February 2023, Longuetaud et al. [36] introduced the $TreeTrace\_Spruce$ database, which contains images and measurements of 100 Norway spruce samples. While both databases are valuable contributions to the forestry community, they lack digital tree-ring annotations for each image, which is needed to assess computer vision algorithms.

The software tool used to generate the annotations was Labelme [59], which produces closed polygons defined by a set of vertices. We worked with two types of annotations: those corresponding to annual ring boundaries or earlywood–latewood boundaries, and those corresponding to reaction wood.

In the case of ring boundary annotations (annual rings and/or earlywood–latewood boundaries), the user placed vertices along the ring borders, following the entire contour. For reaction wood annotations, the user outlined the corresponding region (see Figure 1.3). The tool allows assigning a specific label to each annotated polygon.

Table 2.1: Cross-Section datasets annotations summary. Number of samples (Nb), annual rings (AR) annotations, earlywood-latewood ring boundary (EW-LW), reaction wood (RW), tree or shrub species (Species), and image acquisition method (Acquisition).

Collection	Nb	AR	EW-LW	RW	Species	Acquisition
UruDendro1	64	Yes	Yes	Yes	*P. taeda*	Smartphone
UruDendro2	53	Yes	Yes	Yes	*P. taeda*	Smartphone
UruDendro3	17	Yes	No	No	*G. triacanthos*	Professional Camera
UruDendro4	102	Yes	No	No	*P. taeda*	Smartphone
DiskoIsland	50	Yes	No	No	*S. glauca*	Scanner
Kennel et al. [26]	7	Yes	No	No	*Abies alba*	Scanner

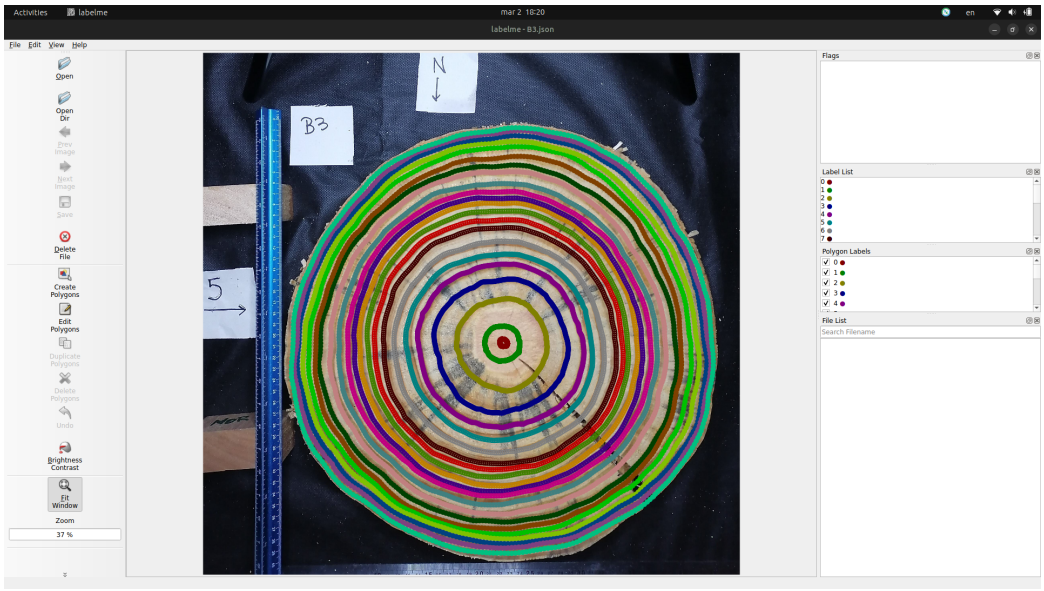


Figure 2.1: Labelme application for annotating tree rings. Rings are represented as polylines in the image.

Vertices can be freely placed anywhere on the image. The graphical user interface (GUI) supports zooming in and out, facilitating the capture of fine image details (see Figure 2.1). Additionally, annotations (each polygon) can be exported in JSON format and later modified by editing the file as needed.

As mentioned earlier, at the beginning of this project, no automatic algorithms for ring delineation were available; therefore, all annotations were performed manually by our team members in the UruDendro1 dataset. The datasets UruDendro3, DiskoIsland, and those of Kennel et al. [26] were also entirely annotated by our team of experts without any automated assistance. Once the CS-TRD method (see Chapter 4) was implemented and functional, it was incorporated into the annotation process. Experts generated the initial ring annotations using this method and subsequently edited them through the GUI as needed. This semi-automatic procedure was applied to the UruDendro2 and UruDendro4 datasets. All annotations are easily visualized over the image sample using the software tool developed during this work (see Chapter 5).

2.2 File structure

In general, the file structure for each collection goes as follows

- images/ - Folder with the image samples
- images_no_background/ - Folder with the images after the background has been removed
- annotations/
  - annual_rings/ - Annual ring annotations
  - earlywood_latewood/ - Earlywood-Latewood boundary annotations
  - reaction_wood/ - Reaction wood annotations
- pith_location.txt - file with the pith pixel location for each image

Where each sample has a unique code. For example, sample F02c, which belongs to the UruDendro1 collection, has five different files:

- images/F02c.jpg
- images_no_background/F02c.jpg
- annotations/annual_rings/F02c.json
- annotations/earlywood-latewood/F02c.json
- annotations/reaction_wood/F02c.json

2.3 Pinus taeda L.

Three collections formed by samples of *P. taeda* species were created. Images from different harvests were acquired and annotated for UruDendro1, UruDendro2, and UruDendro4 collections.

UruDendro1

Harvest Site

Fourteen trees were collected in February 2020 from two tree plantations located in Buena Unión, Rivera, in northeastern Uruguay (Plywood company stands: 31°17’45"S 55°41’42"W; Lumber company stands: 31°10’4"S 55°39’42"W). This region has a humid subtropical climate (*Cfa*, Koppen Climate Classification) with a mean annual rainfall of 1472 ± 373 mm/y (mean ± standard deviation), ranging from 830 to 2797 mm annually and a mean annual temperature of 17.2 ± 0.4 degrees Celsius according to publicly available climate data from the National Institute of Agricultural Research (INIA) Tacuarembo meteorological station concerning the years 1978 to 2022 [24]. Soils in both stands are

(a) (b) (c)

Figure 2.2: Example images from the UruDendro1 dataset. a) Sample F03d. b) Sample F02c. c) Sample L04e.

comprised of sandstone (*areniscas* in Spanish) and pertain to the national soil classes 7.2 and 7.31 according to the National Commission of Agronomic Study of Soils[1] system. Soil type 7.2 is considered a Tacuarembo sandstone with a slope of 10-15 degrees and deep soils with very good drainage. Soil type 7.31 is also a Tacuarembo sandstone with a 6-10 degree slope and good drainage. Prevailing winds are, in order of importance, from the southwest, east, south, and southeast, with an average intensity of 15 to 20 km/h.

Seven trees were harvested from tree plantations managed by a lumber company (denoted with the letter F) and seven by a plywood company (denoted by the letter L). Each company applied different silvicultural practices. Cross-sections between 5 and 20 cm thick were cut from logs at 10, 165, 200, 400, and 435 cm above ground, totaling 64 cross-sections (1 to 5 cross-sections per tree). The image of each cross-section was assigned an identification code comprising the letter of the company (F or L), a two-digit number corresponding to the individual tree, and a lowercase letter corresponding to the height at which each cross-section was cut. Height codes were as follows: a = 10 cm, b = 165 cm, c = 200 cm, d = 400 cm, and e = 435 cm above ground.

Image Acquisition

The cross-sections were dried at room temperature without further preparation. Due to the drying process, radial cracks and blue fungus stains appeared in many samples. Surfaces were polished using a handheld planer and a rotary sander, progressing from 60- to 1000-grit sandpaper. Photographs were taken under different lighting conditions; cross-sections a, b, and e were photographed indoors using an iPhone 6S cell phone (RGB sensor, 12 Mpx), with the surface moistened to maximize the contrast between early and late wood. In contrast, photographs of dry cross-sections c and d were taken outdoors using a Huawei P20 Pro cell phone (RGB sensor, Leica Vario-Summilux, 40 Mpx). For each image, the north side of the tree was manually marked. Figure 2.2 shows some images of the Urudendro1 collection. Minimum image width size is 897 pixels, while maximum image width size is 2736 pixels.

[1] In Spanish *Comisión Nacional de Estudio Agronómico de la Tierra* (CO.N.E.A.T.)

Sample codes follow the structure:

$$XYYZ$$

Where X indicates the collection source (F or L), YY refers to the tree identifier (e.g., 02, 07, 11), and Z to the disc height (a, b, c, d, or e).

Annotation Details

Table 2.1 summarizes the annotation details (see the UruDendro1 row). Annotators labeled annual rings, earlywood-latewood boundary, and reaction wood (see Figure 1.3). All image annotations were revised by more than one expert.

2.3.1 UruDendro2

Harvest Site

The samples were collected in Curticeiras, Department of Rivera, Uruguay, during 2022 (18 samples) and 2023 (35 samples) as part of a harvesting operation managed by a local lumber company. The sampling locations correspond to the following coordinates: 31°17'45" S, 55°41'42" W; 31°10'4" S, 55°39'42" W (18 samples); and 31°01'11.37" S, 55°31'55.81" W (35 samples). In total, 53 cross-section samples were obtained.

The site is classified under the CONEAT 7.31 group and is characterized by being located on the Tacuarembó Sandstone geological formation. It presents slopes ranging from 6% to 10%, very good drainage, and deep soils reaching up to 180–200 cm in the B horizon.

Image Acquisition

The UruDendro1 collection was expanded with 53 new images captured under laboratory conditions using an iPhone 6S (12 MP RGB sensor) positioned at a distance of 43 to 51 cm from the wood cross-section, and illuminated with a 35 W LED ring light. The surface preparation of the samples varied: some were cut using a chainsaw, others were smoothed with a handheld planer, and several were further polished with a rotary sander. All images were acquired indoors. The image widths range from a minimum of 2267 pixels to a maximum of 3024 pixels. Figure 2.3 illustrates representative samples from the dataset.

Sample codes follow the structure:

$$XY$$

Where X denotes the diameter class to which the disc belongs (A, B, or C; see Table 2.2), and Y corresponds to the sample identifier.

Table 2.2: Cross-Section diameter classes for sample categorization.

Class A	**Class B**	**Class C**
35–40 cm	40.1–45 cm	45.1–50 cm

(a) (b) (c)

Figure 2.3: Example images from the UruDendro2 dataset. a) Sample C11C. b) Sample C13C. c) Sample C14C.

Table 2.3: Harvest dates for each treatment and block.

Treatment	Harvest Date
T0	B1, B2 and B3 second week of July
T2	Last week of April
T4	B1 and B2 last week of April; B3 first week of June
T6	B1 and B3 first week of June; B2 second week of May

Annotation Details

Table 2.1 summarizes the annotation details (see the UruDendro2 row). In *P. taeda*, human annotators labeled annual rings, earlywood-latewood boundary, and reaction wood (see Figure 1.3). All image annotations were revised by more than one expert.

2.3.2 UruDendro4

Harvest Site

A total of 102 cross-sections of the species *P. taeda* were collected from the site known as "La Altura", located approximately 4 km from the city of Tranqueras (31°13'30.2" S, 55°46'53.6" W), Department of Rivera, Uruguay. Trees were felled using a chainsaw, and four discs per stem were extracted at heights of 0m, 3m, 6m, and 9m. An additional disc was collected at 1.3m for the T0 treatment. The samples were harvested in 2024, and the specific harvest months are detailed in Table 2.3. Silvicultural treatments are described in Table 2.4.

The soils in the area are very deep, with limited drainage, loamy-sandy in texture, and extremely low fertility, having a productivity index of 53. The dominant soils are Umbric Albic Luvisols. These are classified as priority forestry soils and belong to the CONEAT group 7.42. The terrain consists of gently rolling hills with slopes of up to 3%.

Table 2.4: Description of silvicultural treatments: thinning schedule, tree densities before and after thinning, pruning height, and current stand density.

Treatment	Thinning timing [years]	Pre-thinning density [trees.ha^{-1}]	Post-thinning density [trees.ha^{-1}]	Pruning height [m]	Current density [trees.ha^{-1}]
T0 – Commercial management	11	600	350	5.5–8.0	350
T2 – Heavy thinning / High pruning	8	600	250	8.0	250
T4 – Current thinning / High pruning	8	600	350	8.0	350
T6 – Two thinnings / High pruning	1st: 8 2nd: 18	600	1st: 400 2nd: 200	8.0	200

(a) (b) (c)

Figure 2.4: Example images from the UruDendro3 dataset. a) Sample T0_B1_N27_C. b) Sample T2_B3_N9_C. c) Sample T4_B3_N1_C.

Image Acquisition

The discs were polished using an electric planer. In addition, T0 samples were also sanded using an electric belt sander. At the time of image capture, the discs were moistened. Photographs were taken under varying lighting conditions: samples from T2 and T4 (B1 and B2) were photographed indoors under artificial white light; T6 (B2) samples were photographed outdoors; T4 (B3), T6 (B1 and B3), and T0 samples were photographed outdoors in shaded conditions. All photographs were taken using an iPhone 15 Pro. The minimum image width is 1317 pixels, and the maximum width is 4695 pixels. Figure 2.4 illustrates some examples from the dataset.

Sample codes follow the structure:

$$TX\_BY\_NZ\_W$$

Where TX indicates the treatment applied to the tree (0, 2, 4, or 6); BY refers to the block (1, 2, or 3); NZ refers to the tree identifier (e.g., 3, 12, 23, etc.); and W refers to the sample height (A = between 10 ~30 cm, B = 3m, C = 6m, D = 9m, and ADAP = 1.3m).

Annotation Details

Table 2.1 summarizes the annotation details (see the UruDendro4 row). In the case of *P. taeda*, human annotators labeled annual ring boundaries (see Figure 1.3). Initial ring predictions were generated using the CS-TRD method (Section 4.2) and were subsequently corrected by an expert.

2.4 Gleditsia triacanthos L.

Harvest Site

UruDendro3 is a dataset of seventeen RGB images of *G. triacanthos*, an angiosperm species. Discs were harvested from the different campaigns. UruDendro3a is composed of nine disc samples. In comparison, UruDendro3b is composed of eight disc samples, which were collected from the site known as 'Campo La Nueva Zelanda' (31°48'22.2"S, 56°04'19.2"W), Department of Tacuarembó, Uruguay in 2024. Trees were felled using a chainsaw in both situations.

The soils at the UruDendro3b site belong to the CONEAT group G03.11. These are low plains adjacent to drainage pathways, with virtually flat topography (0% slope), although slight mesorelief may occur. The dominant soils are Melanic Luvic Gleysols (humic gley soils), typically of fine texture and very deep, and Melanic Heterotextural Fluvisols (alluvial soils), which also present variable textures and are very deep.

Flooding can occur for variable periods. The native vegetation typically consists of riparian forest and parkland near watercourses, as well as hydrophilic grassland in more distant areas. In areas with poorer drainage, tall grass wetlands may develop. Land use in this group is mainly limited to summer grazing due to the flood risk.

The subzone G03 corresponds to the Río Tacuarembó unit in the 1:1,000,000 soil map (D.S.F.) and includes the entire plains of the Tacuarembó and Negro rivers and their tributaries. The productivity index of this soil group is 70.

Image Acquisition

UruDendro3a photographs were acquired in a laboratory under uncontrolled illumination conditions, while UruDendro3b photographs were acquired in a laboratory under controlled illumination provided by a 55 W LED lamp. The wood discs were sanded using an electric sander in both cases. However, UruDendro3b samples were photographed while moistened to enhance contrast. Images were captured with a Nikon D3100 camera positioned approximately 50cm from the wood surface. The minimum and maximum image width is 4608 pixels. Sample images are shown in Figure 2.5. Figure 2.6 illustrate sample gttbo2 from the UruDendro3b dataset. The annual ring patterns are more difficult to distinguish compared to those in the *P. taeda* dataset.

Sample code from the UruDendro3a dataset follows the structure *gtX* while sample code from UruDendro3b follows the structure *gttboX*. In both cases, X is an integer identifying the order in which the disc samples were photographed.

Annotation Details

Table 2.1 summarizes the annotation details (see the UruDendro3 row). Human annotators labeled annual ring boundaries. Another expert in the field revised all annotations.

(a) (b) (c)

Figure 2.5: Example images from the UruDendro3a dataset. a) Sample gt11. b) Sample gt6. c) Sample gt8.

(a) (b) (c)

Figure 2.6: Image sample gttbo2 from the UruDendro3b dataset. a) Sample. b) Zoom In. c) Zoom In with annual ring annotation overlapped in green.

2.5 Salix glauca L.

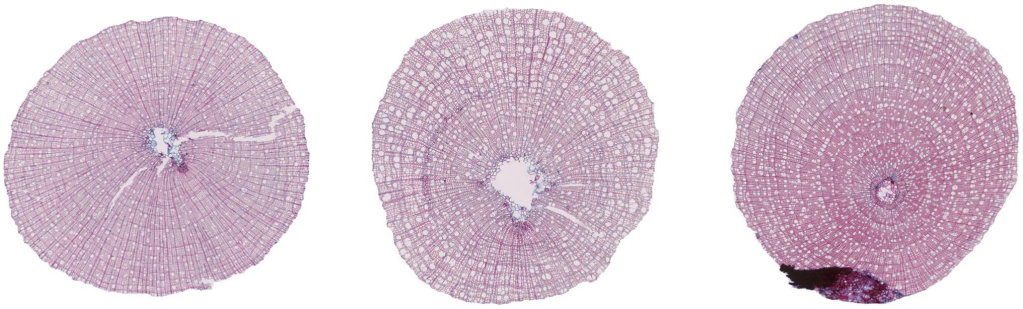
During this thesis, we also collaborated on the tree ring annotation for 50 samples of shrub microscopy images of the *S. glauca* species. This constitutes a new collection, which we have called DiskoIsland.

Harvest Site

Shrubs were sampled in 2019 and 2020 in the Blæsedalen valley on Disko Island (Qeqertarsuaq) (69°16′N, 53°27′ W), in Western Greenland (Kalaallit Nunaat). The study area is in the transition zone between continuous and discontinuous permafrost. The 2020 samples were collected from a snow-fence manipulation experiment conducted on a cut stem (see [49] for more details on the site and sampling process). The 2019 samples were collected using a systematic approach as part of a community-based dendroecological (CBDE) sampling approach (see [48] for more details). The shrubs were sampled between 0- 160 m a.s.l. (i.e., the lowest elevation band in the landscape stratification outlined in [58]). Each systematically selected shrub was excavated for these samples, and the root collar was collected. The samples were cut into 12–15 μm thick microsections with a rotary microtome (RM2245, Leica, Heidelberg, Germany).

Image Acquisition

Shrubs sections were stained with 1 % safranin and 0.5 % astrablue and permanently fixed with Eukitt (BiOptica, Milan). Digital images of each section were captured at 100x magnification (Axio Scan Z1, Zeiss, Germany), resulting in a resolution of 2.26



(a) (b) (c)

Figure 2.7: Example images from the Diskolsland dataset. a) Sample W_F09_T_S2. b) Sample W_F10_C_S3. c) Sample W_F11_C_S1.

(a) (b) (c)

Figure 2.8: Example images from the Kennel dataset. a) Sample 1. b) Sample 4. c) Sample 7.

pixel/μm. The minimum image width is 4472 pixels, and the maximum width is 21692 pixels. Figure 2.7 illustrates some images from the dataset.

Annotation Details

Table 2.1 summarizes the annotation details (see the DiskoIsland row). Human annotators labeled annual ring boundaries. Another expert in the field revised all annotations. In total, 50 image samples were annotated.

2.6 Kennel et al. Abies alba

As the introduction mentions, Kennel et al. [26] published a dataset of seven images of *Abies alba*. Most of these samples are older than the *P. taeda*, having more than 45 tree rings. The minimum image width is 1024 pixels, and the maximum width is 1280 pixels. Figure 2.8 illustrates some image samples from the dataset.

We were unable to process the annual ring annotations correctly. Therefore, tree rings were annotated following the same procedure as the UruDendro1 dataset. Only annual ring boundaries are annotated on this dataset.

2.7 Conclusions

Manual measurement of ring characteristics on wood discs, such as ring width, is costly, as it requires the operator to be physically present with the sample and is time-consuming. According to operator estimates based on manual measurements performed on samples from the UruDendro2 dataset, measuring earlywood and latewood widths in four radial directions took approximately 3 hours per disc. A similar amount of time was reported by annotators for manually delineating annual rings in the images of the *P. taeda* species. For *G. triacanthos*, annotators estimated that it would take up to 5 hours per image to complete the annual ring annotations. In the case of *S. glauca*, the annotation process was estimated to take approximately 1 hour per sample.

Digitizing the discs into images and applying automatic ring delineation techniques can significantly accelerate this process. Once the images are properly acquired, measurements can be carried out from any location, without requiring access to the physical disc. Furthermore, once rings are digitally delineated, additional properties such as growth area and perimeter can be computed automatically.

Creating image datasets with annotated rings is a crucial step in developing automatic detection algorithms. Although time-consuming, these annotations must be produced with high precision and are essential for the training and evaluation of such methods. In species like *G. triacanthos* and *S. glauca*, ring annotation also demands a substantial degree of expertise due to the difficulty in identifying ring boundaries.

All datasets will be publicly available and accessed at: `https://iie.fing.edu.uy/proyectos/madera`.

Chapter 3

Pith detection

This chapter is largely based on the material published in [41] and on the work submitted to the IPOL Journal entitled *Implementation and Discussion of the "Pith Estimation on Rough Log End Images using Local Fourier Spectrum Analysis"*.

3.1 Introduction

Locating the pith of tree cross-sections is essential to identify (in basal discs) the first year of growth and, therefore, the tree's age. The pith has a different type of tissue than the rest of the tree, with distinct physical-mechanical properties. Locating the pith is useful, among other reasons, to detect growing eccentricity because in the natural process of senescence of standing trees, the fungi that degrade the wood enter through the pith or because the industry discards that part as it has different uses than the rest of the wood. Moreover, some tree ring delineation algorithms are sensitive to a precise pith location [4, 17, 39, 44], mainly when those algorithms are based on the ring structure, a concentric pattern similar to a *spider-web* as illustrated in Figure 3.1d. That figure shows some examples of the diversity of images of tree slices. Ideally, the intersection point between the perpendicular lines through the tree rings should be the pith. The *spider web* model is only a general approximation. Real slices include ring asymmetries, cracks, knots, fungus, etc., as seen in Figures 3.1b and 3.1c. Different species produce diverse patterns. Moreover, gymnosperm species, as the ones illustrated in Figure 3.1, and angiosperm ones produce a different wood structure, as seen in Figure 3.9e. Automatic pith detection must be robust to such variations and perturbations.

As other authors (see [4, 39, 44]), we employ the pixel pith model, which is essential for the tree ring delineation method presented in Section 4.2. However, for some species, e.g. young shrubs (under 15 years old), the pith region constitutes a substantial proportion of the cross-section, making it more suitable to model the pith as a region rather than a single point.

This chapter presents several methods to locate the pith on cross sections automatically. In the following sections we present the implementation of the method proposed by Schraml and Uhl et al. [54], the development of two real-time automatic detection methods (APD and APD-PCL), the training of a YoloV8 for this purpose (APD-DL), and

(a) (b) (c) (d)

Figure 3.1: Some examples from *UruDendro1* dataset [40] (a to c). (d) The whole structure, called *spider web*, is formed by a *center* (the slice pith), *rays*, and the *rings* (concentric curves). In the scheme, the *rings* are circles, but in practice, they can be (strongly) deformed as long as they don't intersect another *ring*.

a rigorous comparison of these methods with public implementations of state-of-the-art methods on various datasets. These contributions enhance the field of wood pith detection, offering practical solutions and insights for real-time applications.

3.2 Previous work

Classical algorithms for pith detection follow a three-step approach based on the *spider web* structure defined in Figure 3.1d. First, the normal directions of the ring structures are estimated locally. Second, the intersection point of the lines defined by these normal directions is computed, sometimes using an accumulator space in a Hough Transform-like manner. Finally, a robust estimation of the pith location is extracted from the intersections obtained in the previous step. On this line, Schraml and Uhl et al. [54] proposed a method that splits the wood cross-section into patches, estimating the patch's orientation by 2D Fourier Transform. They accumulate the patch's orientation using a Hough Transform approach and calculate the pith position as the maximum in the accumulation space.

Other Hough-based methods have appeared since Schraml et al. [54] contribution. Kurdthongmee et al. [33] proposed the Histogram Orientation Gradient (*HOG*) to estimate the tree ring local orientations in image patches and then compute their perpendicular orientation as the normal local ring estimation. Local ring orientation is determined by the value of the highest bin of the HOG_i where i indexes the image patch. Intersection points of all the local orientation lines are accumulated, and the point of maximum accumulation is taken as the pith estimation. In the same line, Norell et al. [45] proposed two ways for estimating the local orientations: quadrature filters and a Laplacian pyramids approach. The pith position is estimated by accumulating the local orientations. The accumulation image is smoothed with a Gaussian filter. Then a region-growing approach around the global maxima is used to extract the pith position. Decelle et al. [6] proposed a method based on an ant colony optimization algorithm for the line accumulation step. Firstly, the method runs over the entire image, giving a peak candidate. Then, the procedure is repeated in a square region centered around the first peak candidate. Local orientations are computed using a smooth gradient-based method. Gazo et al. [15] proposed using a

weighted line accumulation step, where the weight increments linearly according to the distance from the outer edge so that the external growth rings have less influence than the internal ones in the final result. They tested their method over computed tomography images.

Deep Neural Network (DNN) methods have also been applied to solve the pith detection problem. Kurdhongmeed et al. [31], compared the effectiveness of two DNN object detector models (YoloV3 and SSD MobileNet) to locate the pith. They trained the models via transfer learning over 345 wood slice RGB images captured within a sawmill environment and used data augmentation techniques. The obtained models were evaluated over a separate dataset of 215 images. Those images present several difficulty levels, from clearly observable annual tree rings to images with annual ring features (highly) degraded by the sawing process. The two DNN models were compared with the Kurdhongmeed et al. [33] pith detection method. The authors claim that a pre-trained SSD MobileNet got the minimum average distance error between all the evaluated methods, performing six times better than a non-DNN method. Additionally, they reported that the pith detection execution time in DNN methods is forty times lower than in non-DNN methods. Furthermore, Gillert et al. [17] proposed using a U-Net architecture trained to segment the pith in microscopy images of shrubs, where the pith is modeled as a region in their work.

3.3 Implementation of the LFSA method

In this section, we analyze the method "Pith Estimation on Rough Log End images using Local Fourier Spectrum Analysis" (here called LFSA) [54], proposed by Rudolf Schraml and Andreas Uhl in 2013, and propose a CPU Python implementation of it.

They proposed two algorithms for pith detection: the first is for RGB images, and the second for tomographic images. Here, we implement the first one, depicted in Algorithm 1. Figure 3.2 illustrates the whole pipeline. The method requires an RGB image of a wood slice and a background mask as input. Figure 3.2a shows the masked slice, i.e., without background. First, in the *local_orientation_estimation* function, the image is divided into blocks, and for each one, a local estimation of the directions of the rings is made. The method also produces a certainty score that measures the confidence in estimating those local dominant orientations. Figure 3.2b illustrates the normals to the local orientation estimation output. Only lines with certainty score higher than a parameter *lo_certainty_th* are kept, i.e., the lines estimated with greater confidence. The closer *lo_certainty_th* is to 1, the better. Then, in the *m_accumulation_space* function, the normals to the dominant direction in each block are traced, and an accumulator space is defined to count the intersections of the normals. Figure 3.2c shows the accumulator space. Finally, in the $find\_peak$ function, the position of the principal peak in the accumulation space is found, which is the pith position. Figure 3.2d illustrates the predicted peak in blue. The global maximums of the accumulator space are colored in red.

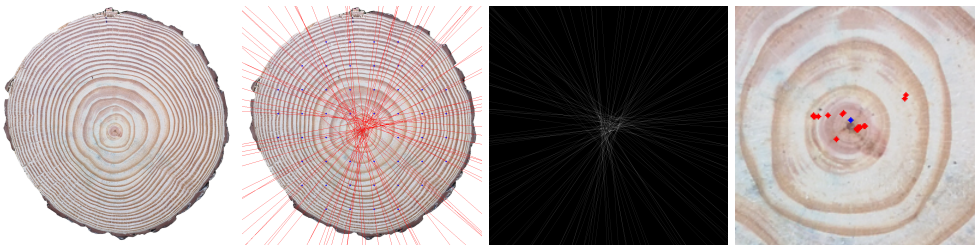
Local Orientation Estimation

The principal hypothesis for locating the pith is that the normals to the local orientation of each ring cross the pith. The image is divided into patches to estimate the local orientation of the tree rings. Inside each patch, the

Algorithm 1: Pith Detection

Input: Im_{in}, // RGB wood cross-section image
mask, // Background mask
Parameters:
block_overlap, // Percentage of overlapping between patches. Patch Stage parameter
block_width_size, // Defines the patch width. Patch Stage parameter
block_height_size, // Defines the patch height. Patch Stage parameter
fft_peak_th, // Frequency filter threshold (λ). Local orientation parameter
lo_method, // Local orientation method. Local orientation parameter
lo_certainty_th, // Linear Symmetry threshold. Local orientation parameter
acc_type, // Lines accumulation method. Accumulator Space parameter
peak_blur_sigma, // Gaussian blurring (σ). Peak parameter
Output: Pixel pith location

1 $l\_lo \leftarrow local\_orientation\_estimation(Im_{in}, mask, block\_overlap, block\_width\_size, block\_height\_size, fft\_peak\_th, lo\_method, lo\_certainty\_th)$
2 $m\_accumulation\_space \leftarrow accumulation\_space(l\_lo, acc\_type)$
3 $peak \leftarrow find\_peak(m\_accumulation\_space, peak\_blur\_sigma)$
4 **return** *peak*



(a) (b) (c) (d)

Figure 3.2: Pith detector pipeline. (a) Masked input image. The background is colored in white. (b) Local Orientation output. The normals to the local patch orientation with good certainty score are drawn. (c) Accumulator Space. (d) Pith's prediction is colored in blue. Accumulator space maximums are colored in red.

rings appear as roughly parallel curves, as shown in figure 3.3a. To estimate their local orientation, the authors use the fact that these patches can be approximated to *simple neighborhoods*, as defined in [19]: a 2D region in which the gray level changes along one direction and is roughly constant along the other. The Fourier transform of a perfect *simple neighborhood* is a line with a similar orientation as the direction of maximal variation of the *simple neighborhood*. As the image departs from a pure 2D sinusoid, this line spreads. The idea is to estimate the orientation of the line, which concentrates the energy in the Fourier transform of the patch. Figure 3.3b shows the spectrum of the patch illustrated in 3.3a. Note how the energy in the Fourier domain is concentrated around a line formed by the two maximums and has the same orientation as the normals to the tree rings. The real image is not exactly a *simple neighborhood*, as the rings are curves instead of lines, and produces a noisy structure in the Fourier transform space that is not exactly a line. However, the principal direction can still be estimated. To facilitate the detection of this direction, the Fourier transform is filtered, as shown in Figure 3.3c. Further details can be read in the reference paper.

Algorithm 2 describes the local orientation detection method. In line 1, the image is split into patches. Three parameters are used:

- *block_overlap*, define the overlapping between patches/blocks. It takes values between 0 and 1. Figure 3.4a and Figure 3.4b illustrate patches with $overlap = 0\%$ and $overlap = 20\%$ respectively,
- *block_width_size*, define the block width size in pixels,
- *block_height_size*, define the block height size in pixels.

The Fourier transform of each patch is computed in line 2, and the Fourier spectrum of the patch is preprocessed in line 3. This stage includes two steps:

1. A band-pass filter is applied with the low frequency defined as $\frac{block\_height\_size}{64}$, and the high frequency defined as $\frac{block\_height\_size}{3}$. These filter parameters are fixed once and for all. Here *block_height_size* is the vertical dimension of the patch.
2. The frequencies lower than fft_peak_th*$\mathbf{M}$, where $\mathbf{M}$ is the maximum magnitude of the patch Fourier Spectrum, are filtered out.

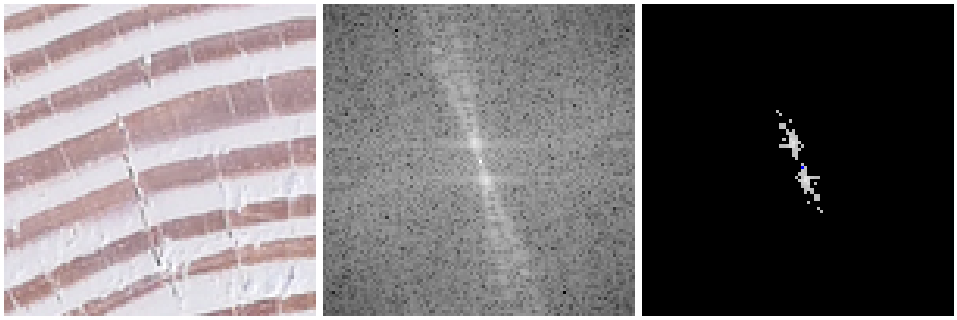
Figure 3.3c illustrates the preprocessed Fourier Spectrum of a patch. In line 4, the local orientation is computed for each patch. Four methods are proposed for the line estimation, each one with a different way to estimate the certainty of the estimation:

- *peak*: The line is defined as the one that connects the frequency with the highest magnitude in the Fourier space with the center. The certainty value is set to 1.
- *lsr*: local orientation is estimated by fitting a line, using a least squares method, for the values of the preprocessed patch Fourier Spectrum (Figure 3.3c). The certainty value is the absolute value of the coefficient of determination $|R^2|$.
- *wlsr*: local orientation is estimated fitting a line by a weighted least squares method for the values of the preprocessed patch Fourier Spectrum (Figure 3.3c). The weights are the square root of the Fourier magnitude. The certainty value is the absolute value of the coefficient of determination $|R^2|$.

Algorithm 2: local_orientation_estimation

Input: Im_{in}, // RGB wood cross-section image
mask, // Background mask
Parameters:
block_overlap, //Percentage of overlapping between patches.
block_width_size, // Defines the patch width.
block_height_size, // Defines the patch height.
fft_peak_th, // Frequency filter threshold.
lo_method, // Local orientation method.
lo_certainty_th, // Linear symmetry threshold. Formulation depends on *lo_method*.
Output: list of lines

1 $l\_blocks, l\_coordinates \leftarrow split\_image\_in\_blocks(Im_{in}, mask, block\_overlap, block\_width\_size, block\_height\_size)$
2 $l\_fs\_blocks \leftarrow compute\_fourier\_spectrum(l\_blocks)$
3 $l\_pre\_fs\_blocks \leftarrow preprocess\_fourier\_spectrum(l\_fs\_blocks, fft\_peak\_th)$
4 $l\_lo \leftarrow lo\_linear\_symmetry(l\_pre\_fs\_blocks, lo\_method)$
5 $l\_lo\_filtered \leftarrow filter\_lo\_by\_certainty(l\_lo, lo\_certainty\_th)$
6 **return** $l\_lo\_filtered$



(a) (b) (c)

Figure 3.3: (a) RGB image patch. (b) The Fourier Spectrum of the image patch shown in (a). (c) Preprocessed Fourier Spectrum of the same patch.

- *pca*: Local orientation is estimated by extracting the orientation of the main direction using Principal Components Analysis. The certainty value is the ratio between the first and second eigenvalues.

Finally, in line 5, the local orientations with a certainty value greater than a *lo_certainty_th* threshold are kept. Figure 3.4 illustrates the local orientation estimation steps.

Accumulator Space The Accumulator Space (AS) is a matrix of the same dimensions as the wood cross-section image. Two types of line accumulations are implemented:

1. Every pixel location where lines are intersected is incremented by one. Lines are represented in the plane by the equation a*x + b*y + c = 0.

2. Every pixel location where a line passes through is incremented by one. We used the function 'line' from the OpenCV Python package to represent a line over an image. This function receives as parameter two pixel locations used as line extremes. In our case, we pass the intersection between the image axis and the line.

Algorithm 3 implements the accumulation space logic. As inputs, it gets the list of lines, *l_lo*. It receives the parameter *type* to indicate which type of line accumulator must be used.

Algorithm 3: accumulation_space

Input: *l_lo*, // list of lines
Parameter:
type, // type of line accumulation.
Output: *m_accumulation_space*, // accumulator space matrix, same dimensions as the image

```
1 if type > 0 then
    /* Increment by one every pixel location where
       two lines are intersected                    */
2   m_accumulation_space ← lines_intersection_accumulation(l_lo)
3 end
4 else
    /* Increment by one every pixel location where
       a line passes through                        */
5   m_accumulation_space ← lines_pass_through_accumulation(l_lo)
6 end
7 return m_accumulation_space
```

Peak estimation A Gaussian filter of kernel size σ, is applied to the accumulator space before predicting the pith location. The predicted pith location corresponds to the global maximum. Their average location is returned if more than one global maximum is found.

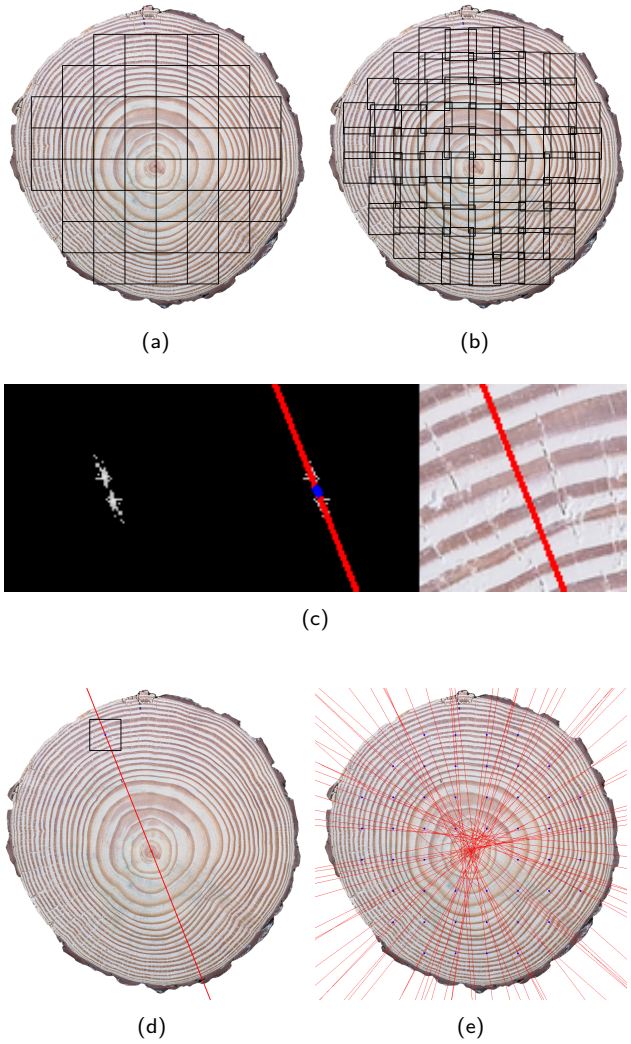


Figure 3.4: Local orientation estimations steps. (a) The wood slice is split into non-overlapping patches. (b) Example of splitting wood cross-section image with 20% overlapping patches. (c) Local estimation of the line. The left image shows the preprocessed 2D Fourier Transform of the patch. The center image shows the line, estimated by the PCA method, over the 2D Fourier Spectrum. The right image shows the estimated line superposed to the patch. (d) A line estimated within a given patch is traced over the whole slice. (e) The output of the local orientation estimation. Only the lines estimated with a certainty score higher than a threshold are traced. Blue dots indicate the patch centers. In this case, the pith location was estimated using non-overlapping patches.

Implementation We implemented the algorithm using Python 3.11. The *README.md* file on the code repository contains all the information needed to run the code. The demo requires, as input, an image of a tree slice.

The program is executed using the following command. As output, the method returns the pixel position of the pith estimation.

```
python main.py --filename IMAGE_PATH  --output_dir OUTPUT_DIR
```

IMAGE_PATH refers to the directory where to search for the wood cross-section RGB image file, and *OUTPUT_DIR* to the path where the output file will be written.

It has the following parameters, which can be set in the command line if needed

1. *–new_shape* One-sided image length after resizing, in pixels. The resizing is always in a square format. The default value is 1000, meaning the resized image is 1000×1000 pixels.
2. *–block_width_size* width block size in pixels (default 100 pixels)
3. *–block_height_size* height block size in pixels (default 100 pixels)
4. *–block_overlap* block overlapping (default 0.2, meaning 20%)
5. *–lo_method* Linear Orientation method (default PCA)
6. *–lo_certainty_th* Threshold to filter out less precise linear orientation (default 0.9)
7. *–fft_peak_th* Threshold (λ) to filter lower frequencies at the Fourier Spectrum stage (default 0.8)
8. *–peak_blur_sigma* Kernel blurring size (σ) applied over the accumulator space
9. *–acc_type* accumulator line logic. If it is set to 1, line intersection logic is applied. If it is 0, every pixel where the line passes through is incremented by 1.

3.4 APD: Automatic Wood Pith Detection

Here we propose an automatic pith detection method based on a model of the wood slice. In a gymnosperm tree cross-section, as the ones shown in Figure 3.1, two types of structures are present: the rings formed by (roughly) concentric curves and, in some cases, the presence of radial structures such as cracks and fungi. Both are fundamentally related to the pith. The former is due to the growing process of the tree, which forms the rings, and the latter is because the tree's anatomy leads naturally to the radial characteristic of cracks and fungus growing. The principal idea of the proposed method derives from this observation: we can locate the pith at the intersection of the lines supported by radial structures and the perpendiculars to the rings.

The angiosperm tree cross-section structure is slightly different, as seen in Figure 3.9e. Still, it is also formed of radially organized cells, with texture patterns appearing at different pith radii. This produces visual macro structures that allow a similar approach to determine the pith position as depicted in the previous paragraph.

Not always do those hypotheses stand out completely. Sometimes, the ring structure can be highly (locally) deformed, as in the presence of a knot. Sometimes, there are no cracks or fungi present. However, in general, enough information is produced by the ring structure and, eventually, by the presence of cracks and fungi to estimate the pith location correctly.

Given an image of the tree cross-section, and using the *spider web* model illustrated in Figure 3.1d, the APD approach pseudo-code is described at Algorithm 4. The main steps are the following (see Figure 3.6 for more details):

Algorithm 4: APD

Input: Im_{in}, // RGB slice image;
Output: Pith location

```
1  ST_O, ST_C ← local_orientation(Im_in, st_σ, st_w)
2  LO_f ← lo_sampling( ST_O, ST_C, lo_w, percent_LO)
3  LO_r ← LO_f
4  for i in 1 to max_iter do
5      if i > 1 then
6          LO_r ← filter_lo_around_c_i(LO_f, r_f, c_i), // See Figure 3.6e
7      c_{i+1} ← optimization(LO_r) //  Equation (3.4)
8      if ||c_{i+1} − c_i||_2 < ϵ then
9          c_i ← c_{i+1}
10         break
11     c_i ← c_{i+1}
12 return c_i
```

1. *Local orientation detection (line 1 of Algorithm 4).* To estimate the local orientation (LO), we compute the 2D-Structure Tensor [2] $ST[p]$ at each pixel p using a window of size $st_w \times st_w$. Pixels in the window are weighted by a Gaussian kernel w of parameter st_σ. The structure tensor is calculated as $ST[p] = \sum_r w[r] ST_{xy}[p-r]$ where $ST_{xy}[p]$ is defined as

$$ST_{xy}[p] = \begin{bmatrix} (I_x[p])^2 & I_x[p]I_y[p] \\ I_x[p]I_y[p] & (I_y[p])^2 \end{bmatrix}$$

where $I_x[p]$ and $I_y[p]$ are the first derivatives of image I in point p along x and y, respectively. We can re-write the 2×2 structure tensor matrix at pixel p as:

$$ST[p] = \begin{bmatrix} J_{11} & J_{12} \\ J_{12} & J_{22} \end{bmatrix}$$

The local orientation at pixel p is:

$$ST_O[p] = \frac{1}{2} \arctan\left(\frac{2J_{12}}{J_{22} - J_{11}}\right) \tag{3.1}$$

The *coherence* of the LO estimation in p is given by the relative value of ST[p] eigenvalues λ_1 and λ_2 (where λ_1 is the largest and λ_2 is the smallest one):

$$STC[p] = \left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}\right)^2 \tag{3.2}$$

The outputs of this step are two matrices: one of local orientations (STO) and one of coherence (STC).

2. *Local orientation sampling (line 2 of Algorithm 4).* The LO estimations are sampled in the following way: 1) STO and STC are divided in non-overlapping patches of size $lo_w \times lo_w$. 2) We find the pixel p^j with the highest coherence (c^j_{high}) within patch $patch_i$. A minimum patch coherence st_{th} is defined. We assign $STO[p^j]$ to $patch_i$ in position p^j, if $c^j_{high} > st_{th}$. To fix st_{th}, we calculate the value of STC such that a given percentage (parameter $percent_{LO}$) of the LO in the slice has $STC > st_{th}$. Each LO is a segment $lo_i = \overline{p^i_1 p^i_2}$, defined by the limits p^i_1 and p^i_2. p^i_{LO} is the middle point between them (p^j). Given the local orientation $\alpha_i = STO[p^i_{LO}]$, points p^i_1 and p^i_2 are computed as $p^i_{1,2} = p^i_{LO} \pm (\cos(\alpha_i), \sin(\alpha_i))$.

   Suppose N patches have coherently enough LO; the output of the step is a matrix, LO_f of size $N \times 4$. In this way, lines are supported by the LO of all meaningful structures in the cross-section, such as the rings. The pseudocode of the step is described in the Appendix A.1.

3. *Find the center (line 7 of Algorithm 4).* Given the filtered local orientation matrix, LO_r, we define the following optimization problem: find c_{opt}, the geometrical position that maximizes the collinearity between the lo_i and a line passing by c_{opt} and p^i_{LO}. To this aim, we define the following cost function:

$$h(x, y) = \frac{1}{N} \sum_{i=1}^{N} \cos^2(\theta_i(x, y)) \tag{3.3}$$

   Figure 3.5 illustrates the vectors involved in computing Equation (3.3). The angle between $\overline{p^i_1 p^i_2}$ and $\overline{c p^i_{LO}}$ is θ_i. The pith position c of coordinates (x,y) is the origin of a segment $\overline{c p^i_c}$. As $\cos(\theta_i(x, y)) = \frac{<\overline{cp^i_{LO}}, \overline{p^i_1 p^i_2}>}{|\overline{cp^i_{LO}}||\overline{p^i_1 p^i_2}|}$, the optimization problem to be solved becomes:

$$\begin{aligned} c_{opt} = \max_{c} \quad & \frac{1}{N} \sum_{i=1}^{N} \left(\frac{< \overline{cp^i_{LO}}, \overline{p^i_1 p^i_2} >}{|\overline{cp^i_{LO}}||\overline{p^i_1 p^i_2}|} \right)^2 \\ \text{s.t.} \quad & c \in Slice\ Region \end{aligned} \tag{3.4}$$

4. To find the global maximum (c_{opt}) of the former optimization problem, we use the *SLSQP*[1] algorithm [30]. Problem 3.4 has a global maximum and is unique if it is

[1] Using the python implementation in *scipy.optimize.minimize* method.

restricted to the region of the wood cross-section. To initialize the *SLSQP* method, we use the least squares solution of finding the point c_{ini}, which minimizes the distance to all the lines in LO_r within the slice.

5. *Refinement (lines 4 to 11 of Algorithm 4)*. Once a candidate for the pith location c_{opt} is obtained, the optimization procedure (3.4) is repeated using only the local orientations within a squared region of size $Size_{image}/r_f$ centered in c_{opt} (see Figure 3.6.e). This step is repeated until the pith location doesn't move more than a given tolerance ($\epsilon = 10^{-5}$) or the iteration counter reaches $max\_iter = 5$. This approach avoids distortions introduced by an asymmetric tree ring growth pattern.

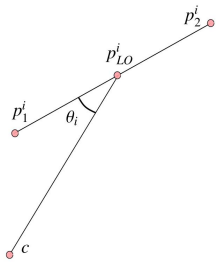


Figure 3.5: Cost function definitions

3.5 APD-PCL: PClines based Automatic Wood Pith Detection

The APD method described in Section 3.4 works fine when the ring structure gives enough information. In some (rare) cases, the ring structure is not visible due to fungi or other perturbations. In those cases, it is possible to solve the same problem using the lines supported by the radial structure of those perturbations and the lines produced by the ring structure. Hence, the APD-PCL version of the method is more robust and allows for the successful treatment of cross-sections with highly degraded ring patterns. The price to pay is a slower algorithm, as it includes a RANSAC-based clustering step.

The APD-PCL method selects which local orientations to consider in the optimization problem of Equation (3.4). In general, the estimation made by the structure tensor calculation step is determined by the rings. Different perturbations also produce some LO, but its number is minimal, and the lines they support don't converge to the pith. In some (rare) cases, the perturbations are so important that they overshadow the ring structure. In those cases, the number of LO produced by the perturbations is more significant than those produced by the ring structure. The set of perturbations-related LO can be of diverse origin: knots, fungi, cracks, and noise. Some of them (as fungi and cracks) have a typical radial orientation, so the perpendicular lines to its LO converge to the pith. Considering this, we modify Algorithm 4, by including a post-processing step over matrix LO_f, between lines 2 and 3. The rest of the algorithm is the same:

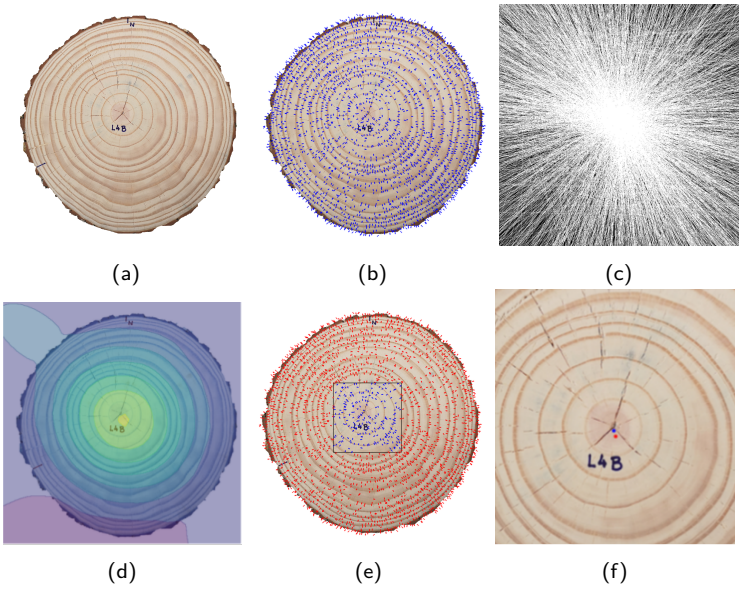


Figure 3.6: Principal steps of APD method (L04d image from UruDendro2 collection). (a) Resized slice image, without background; (b) Sampled LO produced by the Structure Tensor estimation; (c) Accumulation space defined by the LO supported lines; (d) Plot of the cost function (Equation (3.3)), highest values in yellow; (e) Sub image built around the solution c_1 obtained after the first iteration; (f) Evolution of c_i. The final solution is in blue; previous iterations' solutions are in red.

1. Use the PClines transform [8] to convert each line into a point.

2. The PClines space is formed by two sub-spaces defined by a parameter d: the *straight space* includes lines with orientations $\alpha_i \in [0, \frac{\pi}{2}]$ and the *twisted space* lines with orientations $\alpha_i \in [\frac{\pi}{2}, \pi]$. As seen in Figure 3.7, convergent lines in the Euclidean space correspond to aligned points in the PClines spaces. This allows the following steps:

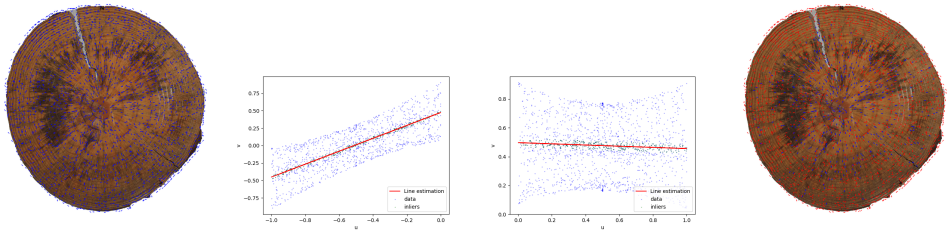
   (a) Lines supported by LO produced by the ring structure converge somewhere around the pith. They produce a line-shaped cluster in the PClines spaces (Figure 3.7b and Figure 3.7c). Working only in the $[-d, 0]$ and $[0, d]$ ranges for the twisted and straight sub-spaces, we select the aligned points using a RANSAC [12] approach. This avoids the use of points near the infinity. We select the converging lines in each sub-spaces, excluding those simultaneously selected in both.

   (b) The previous step clusters all convergent LO_f in the image producing the set LO_{ring}. We rotate by 90 degrees all the orientations in LO_f and repeat the previous procedure to detect the converging ones. These rotated converging lines cluster, LO_{radial}, are produced by cracks, fungi, or similar structures. Adding both gives the set:

$$LO_f^{PClines} = LO_{ring} + LO_{radial}$$

.

(c) To make the line segment selection method more robust, we add a third PClines transform using the lines supported by $LO_f^{PClines}$. Most ring-related LO and rotated LO generated by radial structures are expected to converge (hence, to form a line cluster in the PCline space). Therefore, most of the outliers should be removed at this step.

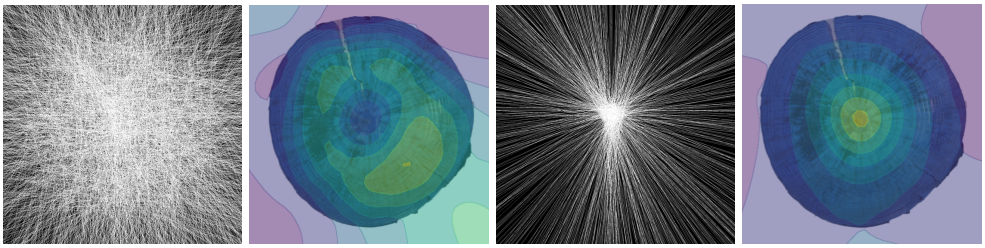


(a) (b) (c) (d)

Figure 3.7: Use of PClines to cluster converging local orientations for slice F07e. (a) Local orientations; (b) Selection of the converging segments in the twisted space using RANSAC to fit a line (in red). Inliers are colored in green; (c) The same procedure is applied in the straight space; (d) In blue, the converging LO (inliers from both sub-spaces) and the LO to be removed in red.

Figure 3.8 illustrates the considered lines and the accumulation space of Equation (3.3) without and with the PClines step. Note how the method filters out many non-convergent lines and regularizes the cost function.

The APD-PCL method is similar to the APD one, but the PClines-based filtering step diminishes the number of considered lines, filtering out many non-convergent ones.



(a) (b) (c) (d)

Figure 3.8: LO Accumulation space and cost function for slice F07e with and without applying the PClines filtering method. (a) LO Accumulation space without filtering; (b) cost function without filtering; (c) LO Accumulation Space with PClines filtering; (d) cost function with PClines filtering.

3.6 APD-DL: Deep Learning based Automatic Wood Pith Detection

In Sections 3.4 and 3.5, we tackle the pith detection problem using a *spider web* model, as in the *classic* image processing approach. Now, we present a Deep-learning approach that learns the model from the data. Inspired by Kurdthongmee et al. [32], who used a YoloV3 model, we train a YoloV8 [25] network using the datasets described in Section 3.7. This is an architecture tailored for object detection and segmentation. To train the model, we need to supply a dataset of wood cross-section images, each labeled with the ground truth pith location indicated by a bounding box, where the bounding box size is one-tenth of the image dimensions.

We divide the data into five sets and use five-fold cross-validation. In each fold i, we use one set ($test_i$) for testing and the other four for training. The training process in each fold is done as usual, and we use the produced model to label the data in $test_i$. The process is repeated for all the folds. In the end, we have predictions for all the data, and in each case, the used model was generated without the influence of the $test_i$ data. With the predictions for all images produced in this manner, we can deliver the metrics to determine the method's performance.

Table 3.1: Prediction results of 5-fold cross-validation for the APD-DL. The second to fourth columns show the Mean (and standard deviation in parenthesis), Median, and Maximum normalized error (defined in Section 3.8.2) values. The last column shows the false negatives. We use all the datasets together to train the model and calculate the performance within each dataset.

Collection	Mean (Std)	Median	Maximum	FN
Uru1+Uru2	0.55 (1.45)	0.18	11.32	2
Uru3a	0.13 (0.06)	0.13	0.27	0
Kennel	0.14 (0.07)	0.13	0.24	0
Forest	0.45 (1.85)	0.12	13.91	0
Logyard	0.52 (1.29)	0.27	7.51	0
Logs	0.22 (0.46)	0.13	4.42	1
Discs	0.23 (0.54)	0.14	5.67	0
All	0.33 (1.01)	0.14	13.91	3

Table 3.1 shows the results using normalized errors (see Section 3.8.2). Training with such a high diversity of data produces state-of-the-art results. Results over each row (collection) are calculated using the predictions and the bounding box center, produced during the five-fold cross-validation with all the images in the seven datasets. In some rare cases, this approach doesn't give a prediction (hence a false negative). In those situations, the method provides the center of the image as the pith position. This explains the relatively large value of the Maximum error and the differences between the Mean and Median errors. Besides the rare false negatives, the results are excellent. The last row depicts the results for all the collections.

Hyperparameters The algorithm was trained with 640 pixels width images (keeping the aspect ratio), using a batch size of 16, for 100 epochs, with the optimizer AdamW [38] (lr = 0.002, $momentum$ = 0.9) and yolov8n as pre-trained weights. All the network was re-trained.

3.7 Datasets description

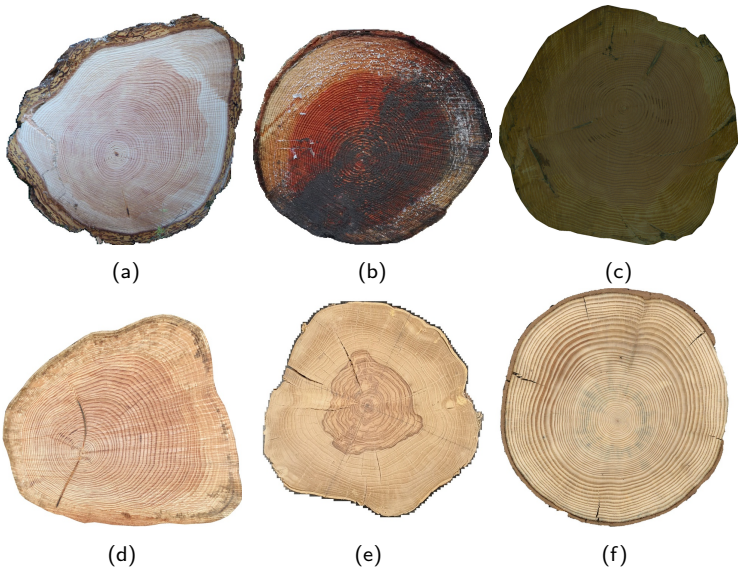


Figure 3.9: Examples of the used datasets. Species are (a-d) *Douglas fir*, (e) *Gleditsia triacanthos*, (f) *Abies alba*. Acquisition conditions: (a-c) in the field, with a smartphone camera; (d-e) in the laboratory, with controlled illumination. The samples (d-e) were previously sanded and polished. Images (a-c) didn't have any special treatment. a) Forest collection. b) Log-yard collection. c) Logs collection. d) Disc collection. e) UruDendro3a collection. f) Kennel collection.

The following datasets are used for evaluation of the pith detection methods:

- *UruDendro* dataset composed by two collections: 117 RGB images of *Pinus taeda* and 9 RGB images of *Gleditsia triacanthos*. Both of them were acquired with a smartphone under laboratory conditions. *Pinus taeda* dataset is composed of the images of UruDendro1 and UruDendro2 in Chapter 2. *Gleditsia triacanthos* dataset is composed of the images of UruDendro3a in Chapter 2.
- Kennel [26]. A public dataset with 7 RGB images of *Abies alba*, polished and acquired in controlled illumination laboratory conditions.
- TreeTrace [35]. It is a public dataset, with samples of *Douglas fir* taken at different stages of the wood process chain. This dataset includes the following collections:
  - Forest, 57 RGB images taken from the freshly cut logs with a digital camera.

- Logyard, 32 RGB images of the same log ends, acquired with a smartphone in the sawmill courtyard several days after the cutting.
- Logs, 150 RGB images acquired in the sawmill with a smartphone.
- Discs, 208 RGB images acquired with a 400 dpi scanner from sanded and polished slices after several weeks of air-drying.

Table 3.2: Datasets description.

Collection	**Number of images**	**Specie**
UruDendro1 + UruDendro2	117	*Pinus taeda*
UruDendro3a	9	*Gleditsia triacanthos*
Kennel	7	*Abies alba*
Forest	57	*Douglas fir*
Logyard	32	*Douglas fir*
Logs	150	*Douglas fir*
Discs	208	*Douglas fir*

Table 3.2 summarize the used datasets. Figure 3.1 show images from the UruDendro2 dataset, and Figure 3.9 show examples from the other collections. These datasets convey a high degree of variability. It includes examples of gymnosperm (*Pinus taeda*, *Abies alba* and *Douglas fir*) as well as angiosperm (*Gleditsia triacanthos*). Acquisition conditions are also diverse, including images obtained with a smartphone in the forest and the sawmill. Samples were acquired in the field with dirt, sap, or saw marks, and others were obtained in controlled illumination conditions in the laboratory from polished samples. The samples include perturbations as fungi, cracks, and knots, as can be seen in Figure 3.1 and sap and saw marks (Figure 3.9b and Figure 3.9d). All have the ground truth position of the pith. Considering all datasets, we work with 580 images.

3.8 Results and discussion

3.8.1 Preprocessing

Sometimes, the images are acquired in the field with a smartphone camera, and one image can contain more than one cross-section. Regardless of the method used, all images are preprocessed in such a way as to standardize the image input:

1. *Background substraction*. Produce a new image limited to one slice. To this aim, when possible, we filter out the background using the mask provided in the datasets. If the mask is not provided, we use a deep learning-based method [50], which uses an U^2Net to segment salient objects.

2. *Resize the image*. This step, which is not strictly necessary, allows us to fix the algorithm's parameters once and for all. All images are resized to 640 pixels width,

Table 3.3: Results on all the datasets. Normalized errors. We show the mean error and the standard deviation between parenthesis. Uru 1+2 stands for the union of UruDendro1 and UruDendro2 collections.

	Uru 1 + 2	**Uru3a**	**Kennel**	**Forest**	**Logyard**	**Logs**	**Discs**
LFSA [54]	1.03 (0.85)	1.46 (0.97)	0.42 (0.18)	0.80 (0.36)	1.02 (0.62)	0.80 (0.46)	0.72 (0.43)
ACO [6]	2.23 (6.64)	4.52 (11.96)	0.2 (0.06)	0.24 (0.24)	0.60 (1.11)	0.46 (0.45)	0.24 (0.35)
APD-PCL	**0.42 (0.34)**	0.74 (0.54)	0.19 (0.10)	0.81 (0.98)	0.82 (0.84)	0.52 (0.47)	0.46 (0.57)
APD	1.02 (2.45)	0.55 (0.30)	**0.14 (0.06)**	**0.22 (0.18)**	**0.35 (0.17)**	0.29 (0.33)	0.26 (0.42)
APD-DL	0.55 (1.45)	**0.13 (0.06)**	0.14 (0.07)	0.45 (1.85)	0.52 (1.29)	**0.22 (0.46)**	**0.23 (0.54)**

respecting the original image's aspect ratio using Lanczos interpolation.[2]

3.8.2 Normalized errors

Given the cross-sections' diverse dimensions, presenting the errors in pixels is not informative. Additionally, not all datasets provide millimeter pixel relations. We use the percentage of the equivalent slice radius. Given a prediction P_i and a Ground Truth GT_i, this error is calculated as follows:

$$Err_i = \frac{100 \times Dist(P_i, GT_i)}{Equivalent\_radii(image_i)}$$

Where $Dist(P_i, GT_i)$ is the Euclidean distance between the prediction and the Ground Truth, in pixels. Remember that the pith is modeled as a point in the image within this work. Therefore, $Dist(P_i, GT_i)$ is the distance between points. $Equivalent\_radii(image_i)$ (in pixels) is half the biggest side of the rectangle that circumscribes the slice.

3.8.3 Experiments

The method to fine-tune the APD-DL method is explained in Section 3.6. To determine the best parameters' values for ACO, LFSA, APD, and APD-PCL, we minimize the average of Euclidean distances between the ground truth and the predictions for all used datasets.

For the APD and APD-PCL methods, the parameters *ransac_outlier_th* (0.03), st_σ(1.2) and r_f(7) were set after experiments over a few images. The fixed values are shown in parentheses. The rest of the parameters, $percent_{LO}$, st_w and lo_w were set searching the minimum over the following grid: $percent_{LO}$ in $[0.3, 0.5, 0.7, 0.9]$, st_w in $[3, 7, 9, 11]$ and lo_w in $[3, 7, 9, 11]$. More details about the parameters are in Appendix A.2.

Inferences were made using an Intel Core i5 10300H workstation with 16GB and a GPU GTX1650 (when needed).

3.8.4 Results

In this section, a performance comparison is made between the mentioned methods. Table 3.3 shows the performance of the proposed methods and two state-of-the-art ones

[2]We impose this restriction due to the GPU memory limitations encountered during the training of the APD-DL method.

[6, 54], over the datasets presented in Section 3.7. To compare different-size wood cross-sections, we use the mean error and standard deviation using normalized errors. The performance of the methods differs for each collection due to its specific characteristics regarding species, acquisition conditions, etc. Note that ACO was developed (and tailored) for the TraceTree collections. Its performance degrades when tried on other species (such as UruDendro collections). LFSA performance is more regular across collections. The three methods proposed in this paper outperform ACO and LFSA on all collections. APD and APD-DL perform better for almost all collections. APD outperforms APD-PCL for all collections except UruDendro2, which has some images with fungi and cracks overshadowing the ring structure. Note that in all the cases, the precision of the pith detection is very high.

Table 3.4: Results of all the methods over the whole set of images, i.e., merging all collections. Normalized errors, number of false negatives, and execution time in milliseconds.

Method	**Mean**	**Median**	**Max**	**FN**	**Time (ms)**
LFSA [54]	0.83	0.72	5.03	0	627
ACO [6]	0.79	0.21	36.39	2	918
APD-PCL	0.52	0.34	4.33	0	2339
APD	0.42	0.19	15.44	0	784
APD-DL	0.33	0.14	13.91	3	209

Table 3.4 compares the performance of all tested methods using the 582 images of all collections. All methods presented in this paper outperform LSFA and ACO methods. The APD performance is surpassed only by the APD-DL method but at the cost of some false negatives: images in which APD-DL didn't find a solution. We can see that APD slightly outperforms the APD-PCL method. This is due to the RANSAC algorithm used to cluster points in the PClines space. When there is no apparent clustering of points around a line, RANSAC tries to fit a line anyway, selecting a wrong set of LO and producing a wrong pith localization. This situation sometimes appears in the TreeTrace dataset. Considering each method's mean processing time per image, we must stress that APD-DL and ACO methods run on GPU, while APD, APD-PCL, and LFSA run on a CPU machine. Note that APD is roughly three times faster than APD-PCL. All in all, it is remarkable that the "classic style" model-based proposed methods (APD and APD-PCL) and a Deep Learning one (APD-DL) have similar performance and execution times, allowing real-time applications with a CPU in the APD and APD-PCL cases. In Appendix A.3, we add showcases illustrating how the different methods work under extreme conditions.

3.9 Conclusions and future work

This chapter addresses the problem of wood pith detection on tree slices using classic image processing and machine learning-based approaches. Both approaches are determined by the characteristics of the data used to tune the algorithm. In search of a more general solution, we use a set of diverse datasets, which spans different species, acquisition conditions, and perturbations (from cracks and knots to saw marks and dirt for images acquired

on the field).

We proposed two real-time methods for tackling the pith detection problem based on a *spider web* model. They have excellent performance and run in real-time on a CPU-based machine, and the model allows a clear comprehension of the approach. The limited number of parameters is understandable and can be fixed once and for all. Additionally, we trained a Yolo V8 architecture for pith detection. It has better (although similar) performance but has some false negatives and is more opaque concerning the meaning of its millions of parameters.

Chapter 4

Tree Ring Detection

This chapter is largely based on the material accepted at ICIAP 2025 “DeepCS-TRD, a Deep Learning-based Cross-Section Tree Ring Detector” and on the work submitted to the IPOL Journal entitled “CS-TRD: a Cross-Sections Tree Ring Detection Method”.

4.1 Introduction

Most of the available methods for dendrometry (measurement of tree growth rings) use images taken from cores (small cylinders crossing all the tree growth rings) instead of complete transverse cross sections. Figure 4.1 illustrates some core images. Using cores for the analysis presents some advantages. The rings are measured on a small portion of the trunk, which can be assumed as a sequence of bands with repetitive contrast, simplifying the image analysis. However, this method provides limited information on annual tree growth because it results in a circular approximation based on a rectangular section. In many cases, ring growth is not uniform along different orientations, leading to significant errors. An application that needs the study of the whole cross-section is when studying the angular homogeneity of the ring-tree pattern trying to detect the so-called compression wood [9] for which the lack of homogeneity in the growing pattern produces differential mechanical properties. On the downside, cross-section analysis implies the felling of the tree and includes the challenge of generating a pattern of (almost) concentric closed curves representing the tree rings. As Figure 4.2 shows, several factors increase the difficulty of the task: wood knots, fungi appearing as black spots with shapes following radial directions, and cracks that can be very wide. Moreover, ring boundaries are a small portion of the disc as seen in Figure 4.3.

This chapter presents a founded method for automatically delineating tree rings on cross-section RGB images. The approach takes advantage of the tree cross-section’s general structure and redundant information on a radial profile for different angles around the tree’s pith.

Section 4.2 briefly reviews previous work in the field. Section 4.3 describes the proposed automatic cross-section tree-ring detection algorithm (CS-TRD). Section 4.4 presents in more detail the algorithms. Section 4.5 presents a modification of the edge detection step in the CS-TRD method, replacing it with a U-Net model trained to seg-

Figure 4.1: Examples of core tree-ring images taken from a dataset with 239 images [11].

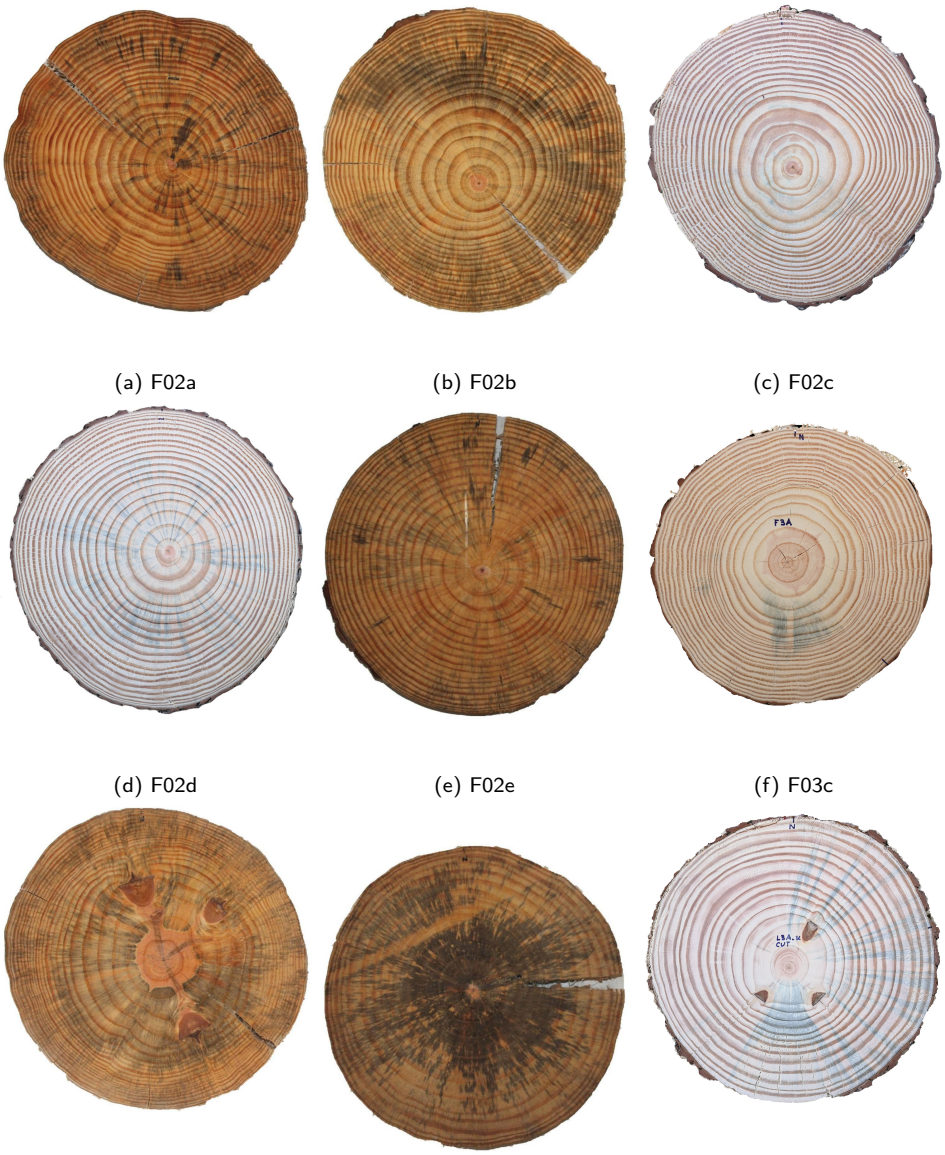
ment annual tree rings. Next, section 4.6 describes the INBD algorithm, a deep learning method originally developed for delineating annual rings in microscopy images of shrubs, which is evaluated in our datasets. Section 4.7 presents the datasets for developing and testing the proposed approach, and section 4.8 discusses experimental results. Section 4.9 concludes and propose future work. Appendix B describes secondary experiments.

4.2 Previous work

Tree ring detection is an old and essential problem in forestry. It has multiple uses in dendrometrology, dendrochronology, ecology, forest management, and other applications. Due to species idiosyncrasies, many practitioners still use a manual approach, using a ruler or other (manual) tree-ring measuring system. This is a tedious and time-consuming task that requires an expert operator.

Among the strategies proposed in recent years for the automation of tree ring delineation over wood cross-section images, we can mention the classic image-processing approaches that try to detect borders and then reconstruct the ring pattern [4, 22, 26, 39, 46, 61] and the algorithms based on deep learning methods which try to learn the solution from the data [17]. The core approaches are more extended, resulting in a significantly greater number of data sets. Hence, the deep learning methods are generally designed for core dendrometry [10, 47].

Cerda et al. [4] proposed a classic image processing approach for detecting entire growth rings based on the Generalized Hough Transform. This work already suggests using the general geometrical structure of the tree rings, which we use in our approach, as illustrated in Figure 4.4. Norell et al. [46] use the Grey Weighted Polar Distance Transform to process end faces acquired in sawmill environments. Still, the method implies using rectangular sections, including the pith, and avoiding knots or other disturbances, diminishing the generality of the approach. Zhou et al. propose a much simpler method [61], which resembles the traditional manual procedure in which two perpendicular lines across the slice are traced. The watershed method is applied to the profiles to obtain the peaks corresponding to each ring. Henkel et al. use an active contours approach [22] alone or coupled with a Dual-Tree Complex Wavelet Transform [26] based on the evolution of a partial differential equation that includes terms related to the image content and to the curve itself. Many PDE-based algorithms are time-consuming, making them difficult to use in real-time applications. Makela et al. [39] proposed a method based on Jacobi Sets to detect the ring pattern and pith location. All these methods rely on detecting the edges corresponding to the tree rings and different strategies to reconstruct the pattern. In all cases, the pith is the center of a general structure. Most of these works were tested against a few images (ranging from 2 to 20). Unfortunately, the code and used images are unavailable in these cases for testing and comparing the results with other methodologies.

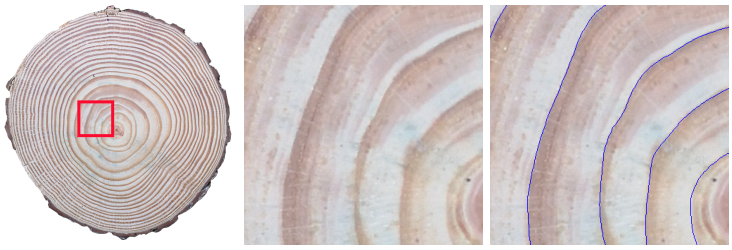


(a) F02a (b) F02b (c) F02c

(d) F02d (e) F02e (f) F03c

(g) F07b (h) L02b (i) L03c

Figure 4.2: Some examples of images from the UruDendro dataset. Note the variability of the images and the presence of fungus (L02b), knots (F07b and L03c), and cracks (F02a, F02e, and L02b). The first five images are from the same tree at different heights, as the text explains in Section 2.3.



(a) (b) (c)

Figure 4.3: Accurate detection of tree ring boundaries in images is critical. (a) Wood cross-section image. (b) Zoomed-in view of the red patch in (a). (c) The same view with ring boundaries in blue. The ring thickness is set to 1 pixel.

Deep learning approaches have become more prevalent in recent years and have naturally been applied to this problem, tangling the ring detection tasks as a segmentation problem. Two architectures are frequent: U-Net and Mask R-CNN [52]. Gillert et al., [17] proposed a method for cross-section tree-ring detection named Iterative Next Boundary Detection Network (INBD) based on the U-NET architecture and applied it to high-resolution microscopic images of shrub cross-sections. The method infers the annual ring at each iteration step, detecting ring by ring from the pith to the tree's bark. INBD was trained and tested on shrub microscopic images. It uses the standard cross-entropy as the main loss for the ring detection network.

Besides the INBD method, most deep-learning-based approaches are applied to core images. Polek et al.[47] used a Mask R-CNN deep learning architecture to detect the rings on cores of coniferous species. During network training, they use the mean average precision (mAP) as a loss function. Fabijańska et al. proposed a classic image-processing approach [11] (based on the linking of maximum image gradient pixels) and a convolutional neural network based on a U-Net architecture [10] for detecting tree rings over core images. Comparing both methods, they reported a significant improvement in precision and recall for the deep learning approach compared to a classic one. The U-Net network was trained to minimize the categorical cross-entropy loss function.

Still, the scarcity of annotated tree ring image datasets is a significant problem in the area, as the methods must be tailored to the particularities of each species. Gillert et al., [17] made available a dataset of 213 high-resolution shrub cross-section annotated images, roughly similar to the *Salix glauca* species (see Figure 4.18d). Kennel et al. [26] made available a dataset of 7 cross-section images of *Abies alba* species, but the ring annotations are not fully accessible. Regarding cores, Polek et al. [47] made available a dataset of 2601 image patches (similar to the red rectangle in Figure 4.3a) with their ring annotations for *Picea abies*, a coniferous.

The INBD method is the only one adapted to treating full cross-section images for which we can access the code, allowing us to compare it with the proposed approach.

In short, existing deep learning-based methods are almost all for cores, and the absence of labeled databases makes it difficult to use them on complete slices. Classical methods are generally based on edge detection and build rings out of them. Most reported methods lack the code or the data to verify their claims. Still, their analysis leads to the

assumption that the presence of perturbations such as fungus, cracks, and knots strongly affects the performance because the construction of each ring depends on the previous ones. Therefore, an error in the pith or center closest rings propagates to the rest of the structure.

4.3 Proposed Approach

In this section, we present the main ideas of the method for tree-ring delineation over RGB cross-section images, called CS-TRD for Cross-Section Tree-Ring Detector.

4.3.1 Assumptions

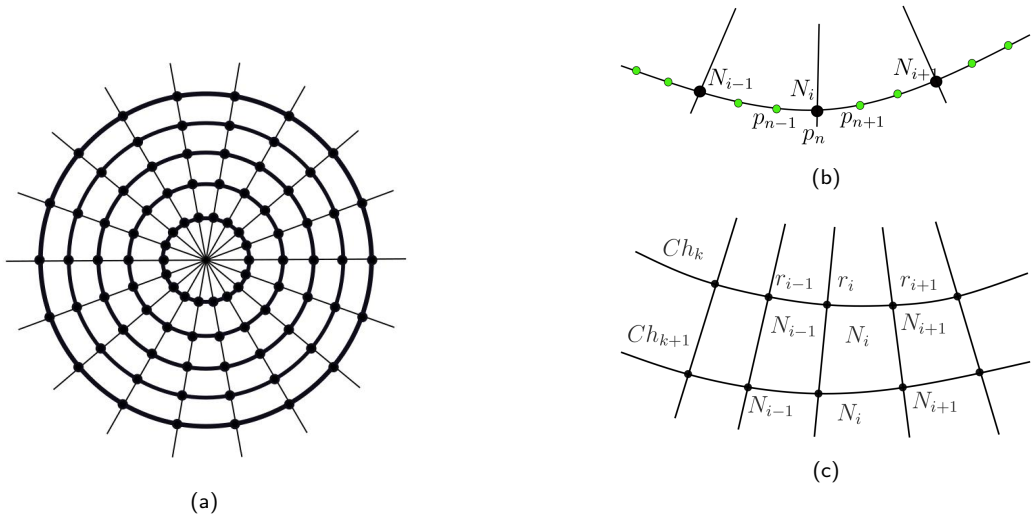
Our tree-ring detection algorithm is heavily based on the structural characteristics of the problem:

- The use of the whole horizontal cross-section of a tree (slice) instead of a wood dowel (or core), as most dendrometry approaches do.
- The following properties generally define the rings on a slice:
    1. The rings are roughly concentric, even if their shape is irregular. This means that two rings can't cross.
    2. Several rays can be traced outwards from the slice pith. Those rays will cross each ring only once.
    3. We are interested only in the rings corresponding to the latewood (darker wood) to earlywood (lighter wood) transitions, namely the *annual rings*.

The principal idea of the method proposed in this work is the definition and use of a general structure formed by the rings, as explained in Section 4.3.2.

4.3.2 Definitions

To explain the approach, we need some naming definitions; see Figure 4.4. We call *spider web* the global structure of the tree-rings we are searching for, depicted in Figure 4.4a. It comprises a *center*, associated with the slice's pith, the origin of several *rays*. The *rings* are concentric and closed curves that do not cross each other. Each *ring* is formed by a *curve* of connected points. Each *ray* crosses a *curve* only once. The *rings* can be viewed as a *curve* of points with *nodes* in the intersection with the *rays*. A *chain* is a set of connected *nodes*. As shown in Figure 4.4b, a *curve* is a set of chained nodes (small green dots in the figure, noted p_n). Depending on the position of the *curve* concerning the *center*, some of those *points* are *nodes* (larger black dots in the figure, denoted N_i hereafter). The larger the number Nr of *rays*, the better the precision of the reconstruction of the *rings*. We fix $Nr = 360$. Note that this is the ideal setting. In actual images, *rings* can disappear without forming a closed curve, the *rings* can be strongly deformed, etc. Figure 4.4.c illustrates the nomenclature used in this paper: *Chains* Ch_k and Ch_{k+1}, intersect the *rays* r_{i-1}, r_i and



(a) (b) (c)

Figure 4.4: (a) The whole structure, called *spider web*, is formed by a *center* (which corresponds to the slice pith), N_r *rays* (in the drawing $N_r = 18$) and the *rings* (concentric curves). In the scheme, the *rings* are circles, but in practice, they can be (strongly) deformed as long as they don't intersect another *ring*. Each ray intersects a ring only once in a point called *node*. (b) A curve is a set of connected *points* (small green dots). Some of those *points* are the intersection with *rays*, named *nodes* (black dots). A chain is a set of connected *nodes*. In this case, the *node* N_i is the *point* p_n. (c) Each *Chain* Ch_k and Ch_{k+1}, intersects the *rays* r_{i-1}, r_i and r_{i+1} in *nodes* N_{i-1}, N_i and N_{i+1}.

r_{i+1} in *nodes* N_{i-1}, N_i and N_{i+1}. The *rays* determine a sampling of the *curves*, producing *chains*, which merge to form a *ring*.

4.4 Algorithm

In this section, the method's main algorithms are described. . Algorithm 5 depicts the CS-TRD method and Figure 4.5 illustrates its intermediate results. The input is an image of a tree disc. First, we subtract the background, applying a deep learning-based approach [50] using a two-level nested U-structure (U^2Net).

Given the image without a background (shown in Figure 4.5b), we must find the set of pixel chains representing the annual rings (dark to clear transitions). We need the *spider web* center $c = (cy, cx)$, the disc's pith, as input. This fundamental point can be manually marked or automatically detected (see Chapter 3). Here, we consider this point given.

4.4.1 Preprocessing

The first step in Algorithm 5 is to preprocess the input image to increase the method's performance. We resize the image to a fixed 1500×1500 pixels size via Lanczos interpolation [16], then convert it to grayscale, and a histogram equalization is applied (Figure 4.5c).

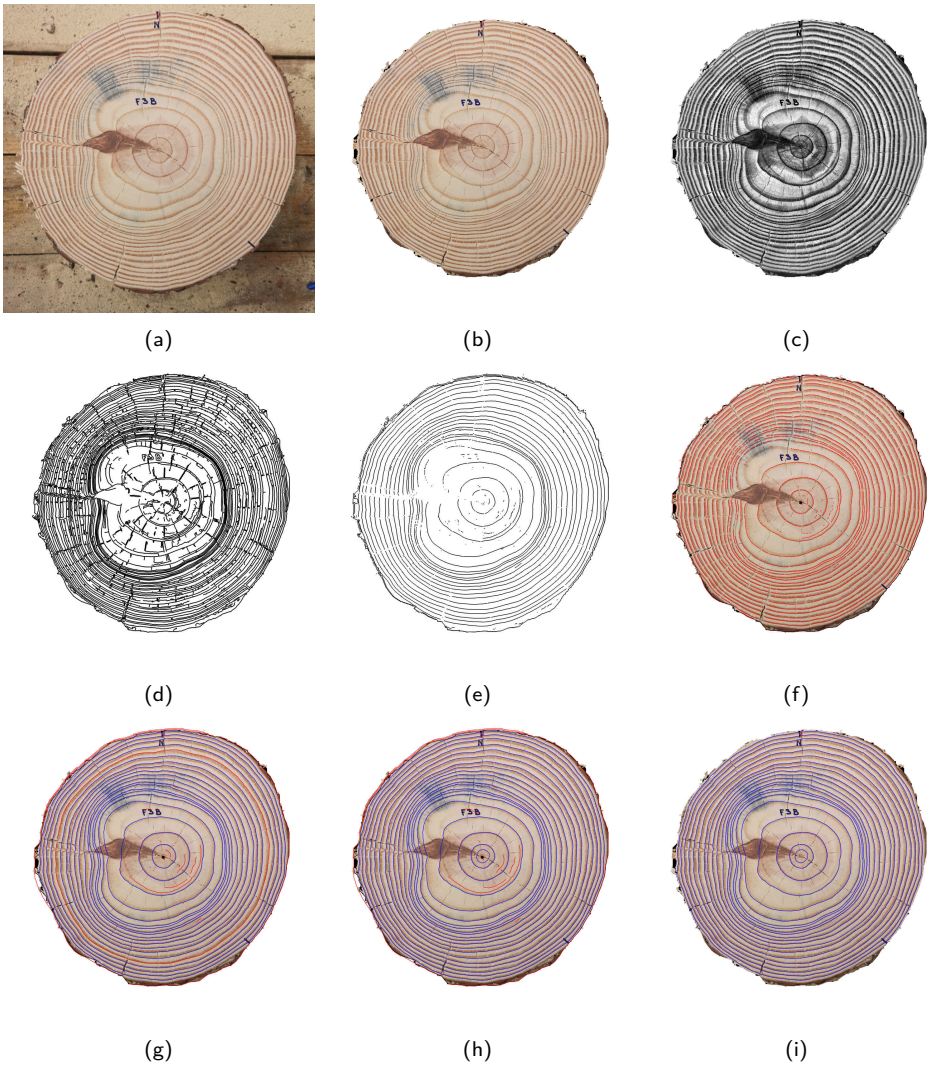


Figure 4.5: Principal steps of the CS-TRD algorithm: (a) Original image, (b) Background subtraction, (c) Pre-processed image (resized, equalized, grayscale conversion), (d) Canny Devernay edge detector, (e) Edges filtered by the direction of the gradient, (f) Detected chains, (g) Connected chains, (h) Post-processed chains and (i) Detected tree-rings.

Algorithm 5: Tree-ring detection algorithm

Input: Im_{in}, // input image
pith_location // pith's coordinates, in pixels (cy, cx)
Output: tree-rings

1 *image* ← preprocessing(Im_{in}) // resizing, grayscale conversion and image equalization
2 *edges* ← canny_devernay_edge_detector(*image*) // use Canny-Devernay detector [21]
3 *edges* ← filter_edges(*edges*, *pith_location*) // preserve edges orthogonal to rays emerging from the pith
4 *chains* ← sampling_edges(*edges*, *pith_location*) // re-sample edges along the rays directions. See Algorithm 6
5 *chains* ← merge_chains(*chains*) // merge neighboring chains and complete them if possible. See Algorithm 7
6 *rings* ← postprocessing(*chains*) // see Algorithm 11
7 **return** *rings*

4.4.2 Canny-Devernay edge detector

Line 2 of Algorithm 5 corresponds to the edge detection stage. We apply the sub-pixel precision Canny Devernay edge detector [7, 21]. The output of this step is a list of pixel chains corresponding to the image edges. Besides some noise-derived ones, we can group those edges into three classes:

- $Edges_T$: produced by the tree growing process. It includes the edges that form the rings. Considering a pith outward direction, they can be of two types: those produced by early wood to late wood transitions (clear to dark) and latewood to early wood transitions (dark to clear). We are interested in detecting the later ones, hereon called annual rings.

- $Edges_R$: radial edges produced by cracks, fungi, or other phenomena.

- Other edges produced by wood knots and noise.

The Canny Devernay edge detector has three parameters: The standard deviation of the Gaussian kernel σ, and the gradient modulus thresholds th_{low} and th_{high}, for the hysteresis filtering of the edge points. We only adjust the parameter σ; the other two are fixed. Figure 4.5d shows the output of this stage. We slightly modified the Canny-Devernay filter implementation [21] to obtain the matrices G_x and G_y, which contain the gradient components in the x and y directions, respectively. These matrices are used in the *filter_edges* step.

4.4.3 Filtering the edge chains

Line 3 of Algorithm 5 corresponds to the edge filtering stage. Given the center $c = (cy, cx)$ and a point p_i of an edge *curve*, the angle $\delta(\vec{cp_i}, \vec{G_{p_i}})$ between vector $\vec{cp_i}$ and gradient vector $\vec{G_{p_i}}$ at p_i is:

$$\delta\left(\vec{cp_i}, \vec{G_{p_i}}\right) = \arccos\left(\frac{\vec{cp_i} \cdot \vec{G_{p_i}}}{||\vec{cp_i}||||\vec{G_{p_i}}||}\right) \tag{4.1}$$

We filter out all *points* p_i for which $\delta(\vec{cp_i}, \vec{G_{p_i}}) \geq \alpha$. We fix $\alpha = 30$ degrees.

Two edge groups are filtered out: earlywood transitions of the $Edges_T$ set gradients (pointing inward), and $Edges_R$ set gradients, roughly normal to the *rays*. The algorithm produces a list of edges, primarily the ring's edges, and the disc perimeter is placed in the last position. Figure 4.5e shows the output of this stage.

4.4.4 Sampling edges

Line 4 of Algorithm 5, is the edge sampling stage which produces the *nodes* depicted in Figure 4.4b. We sample each *curve* of the *edges* list, computing the intersection with the rays defined in Figure 4.4a. Algorithm 6 shows the method's pseudo-code. It has two parameters: N_r (360 by default) and m_c, the minimum number of nodes in a *chain*. As every *chain* has two endpoints, we fix $m_c = 2$. The algorithm produces a list *chains* (of **Chain** objects). List include two artificial *chains*, one of type *center* with N_r nodes and the exact pith coordinates but different angular orientations, and one corresponding to the border (disc's perimeter). These artificial chains are advantageous for the connecting chains stage, discussed in the next section, as they impose an inward and outward boundary structure (see the Appendix B for more details). The attribute *type* of the **Chain** object indicates whether it is an artificial or standard chain.

The rays defined in Figure 4.4a (*l_rays*) are computed in line 1 of Algorithm 6. Edge curves *edges* are sampled in function *intersections_between_rays_and_devernay_curves* (line 2). To do so, we calculate the intersection between each curve and each ray in *l_rays* using the *intersection* method of the *shapely* library [18]. Finally, in line 4, artificial chains are added to the chain list.

Figure 4.5f shows the output of this step. Standard chains are in red, and center and border chains are in black. Due to the sampling, there are fewer chains than edges. Figure 4.5e has more (noisy) curves around the pith than Figure 4.5f. Some small curves with sampled lengths shorter than m_c are discarded. In that sense, this parameter filters out "short" chains.

Every chain has two endpoint nodes. Endpoint A is always the furthest node clockwise, while endpoint B is the most distant node counterclockwise. Given an endpoint, a chain has two attributes: *outward* and *inward* chains. Given the corresponding *ray* of a *chain* endpoint, we find the first *chain* that intersects it going from the chain to the center along the *ray* (named *inward*) and the first *chain* that intersects that *ray* moving away from the center (named *outward*). In Figure 4.6, chains are superposed over the gray-level image. The ray at endpoint A is blue, and the nodes are red. The *visible* chains for the black chain at endpoint A are in orange (outward) and yellow (inward).

Algorithm 6: Sampling Edges

Input: *edges*, // list of *curves*
pith_location, // pith coordinates in pixels: center of the *spider web*
Parameters:
m_c, // minimum length of a chain
Nr // number of total rays
Output: A list *chains* where each element is a *chain*

1 *l_rays* ← build_rays(*Nr*, *pith_location*)
2 *chains* ← intersections_between_rays_and_devernay_curves(*pith_location*, *l_rays* , *edges*, m_c, *Nr*)
3 *pith_chain*, *border_chain* ← generate_virtual_chains(*pith_location*, *Nr*, *chains*)
4 *chains* ← *chains* ∪ *pith_chain* ∪ *border_chain*
5 **return** *chains*

Given $EndPoint_j$ for the current chain Ch_j, and $EndPoint_k$ for chain Ch_k, we can define:

- **Euclidean distance**

$$d_e = \sqrt{\left(x_j - x_k\right)^2 + \left(y_j - y_k\right)^2} \tag{4.2}$$

  Where (x_j, y_j) and (x_k, y_k) are the cartesian coordinates of $EndPoint_j$ and $EndPoint_k$.

- **Radial Difference distance**

$$d_{rd} = \|r_j - r_k\| \tag{4.3}$$

  r_j is the Euclidean distance between $EndPoint_j$ and the pith center, and r_k is the Euclidean distance between $EndPoint_k$ and the pith center.

- **Angular distance**

$$d_a = \left(\theta_j - \theta_k + 360\right) mod \;\; 360 \tag{4.4}$$

  Where θ_j and θ_k are the ray's direction supporting $EndPoint_j$ and $EndPoint_k$ (both in degrees), respectively, *mod* refers to the modulus operation.

4.4.5 Merge chains

We must now group the chains to form rings (Line 5 of Algorithm 5). As this section is a key part of the algorithm, we will divide it into two subsections: one that explains the general idea of merging chains and another that details the algorithms themselves.

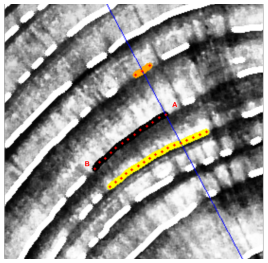


Figure 4.6: A given chain (in black) with two endpoints A and B. Its nodes (in red) appear at the intersection between the Canny Devernay curve and the rays. The ray at endpoint A is in blue. Other chains detected by Canny Devernay are in white. Endpoint A's inward and outward chains are in yellow and orange, respectively.

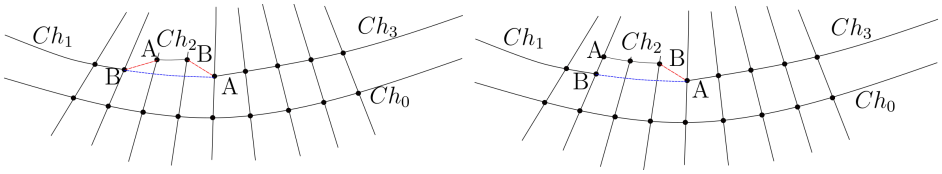


Figure 4.7: An illustration of the *connectivity* issue. (a) The question is if endpoint A of Ch_3 must be connected to endpoint B of Ch_2 (red dashed line) or to endpoint B of Ch_1 (blue dashed line). In figure (b), the same question can be posed for the connection between endpoint B of Ch_1 and endpoint A of Ch_2. This connection is forbidden because Ch_1 and Ch_2 intersect (the endpoints are on the same *ray*). Note that we represent the connections by line segments for clarity, but these are curves in the image space, as we interpolate between *chain* endpoints in polar geometry. Each node is defined by an angle and radial distance from the pith (see Equation (4.3)). Given the endpoints of the chains to be connected, we perform a linear interpolation in polar space to link the chains. This approach ensures that the newly formed chains intersect with a radial line, allowing each connection to have a node at the radial intersection.

General Logic of Chain Merging

The input of the algorithm is a set of chains. Some of these chains are spurious, produced by noise, small cracks, knots, etc., but most are part of the desired rings, as seen in Figure 4.5f.

In general, a ring is composed of multiple chains. To merge them, we must determine whether the endpoints of two given chains can be connected, as illustrated in Figure 4.7a. A support chain (denoted as Ch_0 in the figure) is used to decide whether the chains should be merged. In this context, we use the terms "connect chains" and "merge chains" interchangeably.

To group chains that belong to the same ring, we proceed as follows:

1. We order the chains by length and begin by processing the longest, called *Chain support*, Ch_i. Once we finish merging all the possible *candidate chains* related to that one, we do the same with the next longest *chain*, and so on. The reason for processing the longest chains first is that longer chains contain more information

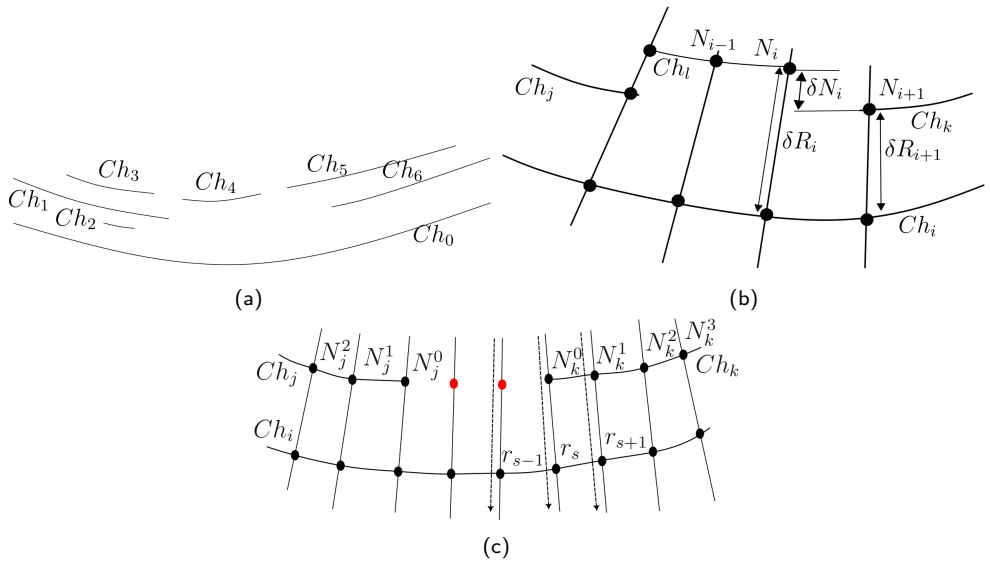


Figure 4.8: Connectivity nomenclature. (a) For the *chain support* Ch_0, the set of *chain candidates* is formed by Ch_1, Ch_2, Ch_4, Ch_5 and Ch_6. *Chain* Ch_3 is shadowed by Ch_1 but Ch_5 is not shadowed by Ch_6 because at least one endpoint of Ch_5 is visible from Ch_0. Note that a *chain* becomes part of the *candidate chains set* if at least one of its endpoints is visible from the *chain support*. (b) Quantities used to measure the connectivity between *chains*. δR_i is the radial difference between two successive *chains* along a *ray* r_i and δN_i is the radial difference between two successive *nodes* N_i and N_{i+1}. Note that these nodes can be part of the same *chain* or be part of two different *chains* that may be merged. Ch_i is the support chain. Ch_i visible chains are Ch_j, Ch_l and Ch_k. Chains Ch_j and Ch_k satisfy similarity conditions. (c) Chains Ch_j and Ch_k are candidates to be connected, and Ch_i is the support chain. N_j^n (N_k^n) are the nodes of Ch_j (Ch_k), with $n = 0$ for the endpoint to be connected, and r_s represents the radial distance to the pith. In red are the nodes created by an interpolation process between both endpoints.

.

about the tree ring structure, giving more robust results. Longer chains are more likely to be an edge belonging to an annual tree ring. Since consecutive rings have similar shapes, this iteration method propagates the shape of the longer chains.

2. We find the chains that are visible from the *Chain support* inwards (i.e., in the direction from *Chain support* to the center). Here, *visible* means that a *ray* that goes through one *candidate chain* endpoint crosses the *chain support* without crossing any other *chains* in between. The set of *candidate chains* of the *Chain support* Ch_i is named $candidates_{Ch_i}$.

   In Figure 4.8a this set is $candidates_{Ch_0} = \{Ch_1, Ch_2, Ch_4, Ch_5, Ch_6\}$. *Chain* Ch_3 is shadowed by Ch_1 and Ch_5 is not shadowed by Ch_6 because at least one of its endpoints are visible from Ch_0. The same process is made for the chains visible from the *Chain support* outwards.

3. We explore the set $candidates_{Ch_i}$ searching for connections between them. By

construction, the *chain support* is not a candidate to be merged in this step. From the endpoint of a chain, we move clockwise or counterclockwise depending on whether the endpoint is A or B. The next endpoint of a nonintersecting *chain* in $candidates_{Ch_i}$ is a candidate to be connected to the first one. We say that two *chains* intersect if there exists at least one *ray* that crosses both *chains*. For example, in Figure 4.8a, Ch_6 intersects with Ch_5 but not with Ch_4.

4. To decide if both chains must be connected, we must measure the *connectivity goodness* between them, combining four criteria:

   (a) *Radial tolerance for connecting chains*. The radial difference between the distance from each chain to be merged (measured at the endpoint to be connected) and the support chain must be small. For example, in Figure 4.8b, if we want to connect node N_i of Ch_l and node N_{i+1} of Ch_k, we must verify that

   $$\delta R_i * (1 - Th_{Radial\_tolerance}) \leq \delta R_{i+1} \leq \delta R_i * (1 + Th_{Radial\_tolerance})$$

   Where $Th_{Radial\_tolerance}$ is a parameter of the algorithm. We call this condition *RadialTol*.

   (b) *Similar radial distances of nodes in both chains*. For each chain, we define a set of nodes. For the chain Ch_j, this set is $N_j = \{N_j^0, N_j^1, ..., N_j^{n_{nodes}}\}$ where n_{nodes} is the number of nodes to be considered, fixed to $n_{nodes} = 20$. See Figure 4.8c. We use the whole chain if it is shorter than n_{nodes}. We measure δR_i, the radial distance between each node in the given chain and the corresponding node (same ray) in the support chain, as illustrated in Figure 4.8b. This defines a set for each considered chain j and k: $Set_j = \{\delta R_j^0, .., \delta R_j^{n_{nodes}}\}$ and $Set_k = \{\delta R_k^0, .., \delta R_k^{n_{nodes}}\}$. We calculate their mean and standard deviation $Set_j(\mu_j, \sigma_j)$ and $Set_k(\mu_k, \sigma_k)$. This defines a range of radial distances associated with each chain: $Range_j = (\mu_j - Th_{Distribution\_size} * \sigma_j, \mu_j + Th_{Distribution\_size} * \sigma_j)$ and $Range_k = (\mu_k - Th_{Distribution\_size} * \sigma_k, \mu_k + Th_{Distribution\_size} * \sigma_k)$, where $Th_{Distribution\_size}$ is a parameter. There must be a non-null intersection between both distributions to connect both chains: $Range_j \cap Range_k \neq 0$. We call this condition *SimilarRadialDist*.

   (c) *Regularity of the derivative*. Given two chains Ch_j and Ch_k which can be connected, let's call Ch_{jk} the set of interpolated nodes between them (Figure 4.8c). The new virtual chain created by the connection between Ch_j and Ch_k will encompass the nodes of those two chains and Ch_{jk}. The parameter *derivFromCenter* controls how are estimated the interpolated nodes between two chains, as the ones in red in Figure 4.8c. If $derivFromCenter = 1$, ray angle and radial distance from the center are used to estimate the position of the interpolated nodes. If its value is 0, the estimation is made by measuring the radial distance to the support chain. To test the regularity of the derivative, we define a set of nodes for each concerned chain. For the chain Ch_j, this set is $\{N_j^0, N_j^1, ..., N_j^{n_{nodes}}\}$ where $n_{nodes} = 20$ is the number of nodes to be considered. If the chain is shorter, we use all nodes. We compute the centered derivative in each node for all chains, $\delta N^s = \frac{\|r_{s+1} - r_{s-1}\|}{2}$, where r_s

Table 4.1: Connectivity Parameters. Each column is the parameter set used on that iteration.

	1	2	3	4	5	6	7	8	9
$Th_{Radial\_tolerance}$	0.1	0.2	0.1	0.2	0.1	0.2	0.1	0.2	0.2
$Th_{Distribution\_size}$	2	2	3	3	3	3	2	3	3
$Th_{Regular\_derivative}$	1.5	1.5	1.5	1.5	1.5	1.5	2	2	2
NeighbourhoodSize	10	10	22	22	45	45	22	45	45
derivFromCenter	0	0	0	0	0	0	1	1	1

is the radial distance of the node N^s to the center (i.e., the Euclidean distance between the node and the center). The set of the existing chains nodes is $Der(Ch_j, Ch_k) = \{\delta N_j^0, ..., \delta N_j^{n_{nodes}}, \delta N_k^0, ..., \delta N_k^{n_{nodes}}\}$. The condition *RegularDeriv* is asserted if the greatest derivative in the interpolated chain is less or equal to the greater derivative in the neighboring chains, with tolerance $Th_{Regular\_derivative}$:

$$max(Der(Ch_{jk})) \leq max(Der(Ch_j, Ch_k)) \times Th_{Regular\_derivative}$$

(d) *Non-Overlapping Chain*. Finally, no other chain must exist between the chains to be connected. If another chain exists in between, it must be connected to the closer one. For example, in Figure 4.8a, it is impossible to connect chains Ch_3 and Ch_5 because between them appear Ch_4. We call this condition *ExistChainOverlapping*.

Summarizing, in order to connect chains Ch_j and Ch_k, the following condition must be met:

$$\textbf{not}ExistChainOverlapping \wedge RegularDeriv \wedge (SimilarRadialDist \vee RadialTol) \tag{4.5}$$

where $\vee$ and $\wedge$ stands for the logical *or* and *and* symbols, respectively, and the symbol **not** stands for the *not* operator.

The method iterates, searching for connectivity between chains over different neighborhood sizes. The parameter *NeighbourhoodSize* defines the maximum allowed distance, measured in degrees, for connecting two chains. We iterate this process for the whole image nine times. In the first iteration, there are a lot of small chains, but as the iterations advance, the concerned chains are already more extended and less noisy. As the merging process advances, we relax the parameters to connect more robust chains. Table 4.1 summarizes the parameters used in each iteration.

5. We proceed in the same manner in the outward direction.

Algorithms description

Algorithm 7 describes the pseudo-code for the *merge_chains* method (line 5 in Algorithm 5). As previously described, the method iterates over nine parameter combinations

(see Table 4.1), which is reflected in the for-loop from lines 1 to 16. For each parameter combination (represented by the variable *iteration*), the algorithm iterates over all chains in the list `chains`, using each one as a support chain to attempt merging with other chains, as described earlier. Once a full iteration is completed, the process continues with the next parameter set. This procedure is shown between lines 2 and 12, where Ch_i denotes the support chain in the current iteration.

Given a support chain, the first step is to identify the visible chains (line 4). These are determined in both outward and inward directions and stored in the lists `l_s_outward` and `l_s_inward`, respectively. The merging process iterates over both sets (line 5). The exploration of candidate chains, as detailed in Item 3, is performed within the for-loop from lines 6 to 11. For a given candidate chain Ch_j from the set $candidates_{Ch_i}$, the algorithm seeks a chain Ch_k to merge with. The first step (line 7) involves selecting the subset of non-intersecting chains concerning Ch_j from $candidates_{Ch_i}$; these are grouped in the list `valid_chains`. Among these, the algorithm selects the one with the smallest angular difference relative to Ch_j, within an angular neighborhood defined by the parameter `NeighbourhoodSize` (see Table 4.1), which corresponds to the current *iteration*. This selected chain is referred to as Ch_k. The endpoint of Ch_j to be connected is denoted as E_j.

If both chains satisfy the conditions specified in Equation (4.5) (line 9), they are merged (line 10). After the merge, Ch_k is removed from the `chains` list, and the updated version of Ch_j replaces the original in the list (line 11).

Once a complete iteration over `chains` for a given parameter set (*iteration*) has been performed, the algorithm exits the main while-loop (lines 3 to 12). As a final step in the *iteration*, the algorithm reviews all chains and completes those that contain more than $0.9 \times Nr$ nodes, a fixed threshold. To complete a chain, the algorithm finds the closest radial chain common to both of its endpoints (Ch_i) and calls the `close_chain` method (see Algorithm 9). Completing a chain involves adding new nodes via linear interpolation (see Line 4) until the chain reaches Nr nodes.

Algorithm 8 describes the logic for connecting chains Ch_j and Ch_k, given a support chain Ch_i. Optionally, chains can also be connected using two support chains, as discussed in the following section (see Section 4.4.6).

First, given the endpoint E_j of chain Ch_j, the corresponding endpoint E_k of chain Ch_k is identified (line 1). Lines 2 to 6 describe the logic for generating new nodes based on one or two support chains. This procedure is detailed in Line 4. Finally, in line 8, a new chain is created by concatenating the nodes from Ch_j, the interpolated nodes, and the nodes from Ch_k.

Algorithm 9 describes the logic for completing the nodes of a chain using either one support chain (Ch_i) or two (Ch_i and $support2$). First, the endpoints of the target chain (E_A and E_B) are obtained. Then, new nodes are generated via linear interpolation, following the same procedure used in Algorithm 8. Finally, if no overlapping chain exists (see Algorithm 10), the chain is completed.

Algorithm 10 describes another key method, corresponding to the term $ExistChainOverlapping$[1] in Equation (4.5). This method checks whether any chains overlap with a given band (see Figure 4.9). An overlapping chain is defined as any chain

[1]The remaining terms in the equation involve straightforward calculations.

Algorithm 7: Merge Chains

Input: *chains*
Output: *chains*

```
1  for iteration in 1 to 9 do
2      Ch_i ← find_support_chain(chains)
3      while Ch_i do
4          l_s_outward, l_s_inward ← find_visible_chains(chains, Ch_i)
5          for candidates_{Ch_i} in [l_s_outward, l_s_inward] do
6              for Ch_j in candidates_{Ch_i} do
7                  valid_chains ← find_non_intersection(candidates_{Ch_i},
                    Ch_j)
8                  Ch_k, E_j ← find_closest(valid_chains, Ch_j, iteration)
9                  if connectivity_goodness_condition( Ch_j, Ch_k, iteration)
                    then
                       /* Equation (4.5) is satisfied     */
10                     Ch_j ← merge_two_chains(Ch_j, Ch_k, E_j, Ch_i) // see
                        Algorithm 8
11                     chains ← update_chains_list( chains, Ch_j, Ch_k)
12         Ch_i ← find_support_chain(chains)
13     for chain in chains do
14         if chain_nodes(chain) ≥ 0.9 × Nr then
15             Ch_i ← get_common_chain_to_both_borders(chain)
16             close_chain(chain, Ch_i) // see Algorithm 9
17 return chains
```

that has at least one node within the band that does not belong to either Ch_j or Ch_k.

The band itself is defined by a node list, *nodes*, which includes the (interpolated) red nodes as well as the endpoints of chains Ch_j and Ch_k (Figure 4.9). This band is constructed using the *InfoVirtualBand* class. The width of the band, denoted as *band_width*, is defined as a percentage of the radial distance to the support chain Ch_i. If Ch_i is of type *center*, then *band_width* = 5%; otherwise, it is set to 10%. These values are fixed.

The algorithm iterates over *nodes*, and for each node it generates two new nodes located on the same ray but at different radial distances from the pith, as shown in Figure 4.9. For a given node $N_i \in l\_nodes$, the radial distances of the corresponding band nodes are computed as $R\left(N_i^{green/red}\right) \leftarrow \delta R_i * (1 \pm band\_width) + R(N_i)$ where $R(\cdot)$ denotes the radial distance to the pith, as defined in Equation (4.3). The nodes N_i, N_i^{green}, and N_i^{red} lie along the same ray. The green and blue nodes are stored in the `info_band` object.

The function `exist_chain_in_band_logic` (line 2) returns the list of chains that intersect the band defined by `info_band`. This is done by iterating over each chain

Algorithm 8: Merge Two Chains

Input: Ch_j, // current chain to be connected
Ch_k, // closest chain to be connected with Ch_j
E_j, //Ch_j endpoint to be connected
Ch_i, //support chain Ch_i
$support2$ // Optional. Second support chain.
Output: merged chain

$E_k \leftarrow$ get_opposite_endpoint(Ch_j,E_j, Ch_k)
if *support2* **then**
    interpolated $\leftarrow$ interpolate_nodes(E_j, E_k, Ch_i. $support2$)
end
else
    interpolated $\leftarrow$ interpolate_nodes(E_j, E_k, Ch_i)
end
merged $\leftarrow$ $Ch_j \cup interpolated \cup Ch_k$
return *merged*

Algorithm 9: Close Chain

Input: *chain*, // Current chain to be connected
Ch_i, // Support chain Ch_i
$support2$, // Optional. Second support chain
Output: closed chain

E_A, $E_B \leftarrow$ get_chain_endpoints(*chain*) //locate nodes belonging to non-angularly-consecutive rays
if *support2* **then**
    interpolated $\leftarrow$ interpolate_nodes(E_A, E_B, Ch_i. $support2$)
end
else
    interpolated $\leftarrow$ interpolate_nodes(E_A, E_B, Ch_i)
end
`/* See Algorithm 10 */`
if *not* *exit_chain_overlapping*($E_A \cup interpolated \cup E_B$, *chain, chain,* Ch_i) **then**
    closed_chain $\leftarrow$ *chain* $\cup$ *interpolated*
end
else
    closed_chain $\leftarrow$ *chain*
end
return *closed_chain*

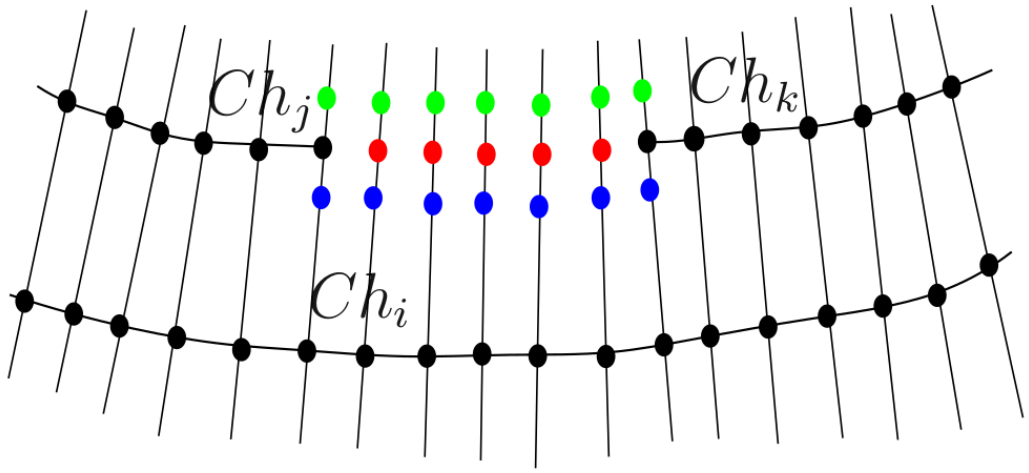


Figure 4.9: Red nodes are the interpolated ones between Ch_j and Ch_k chains. Blue nodes define the outer band (outward), while green nodes define the inward band. Ch_i is the support chain.

and checking if any of its nodes fall between the corresponding blue and green band nodes. Chains that intersect the band are added to the list `l_chains_in_band`. If the length of this list is greater than zero, it indicates that at least one overlapping chain exists within the specified band.

Algorithm 10: Exist Chain Overlapping

Input: *nodes*, // list of interpolated nodes plus the endpoints
Ch_j, // source chain. Check Figure 4.9
Ch_k, // destination chain. Check Figure 4.9
Ch_i, // support chain
Output: Boolean. True if exists a chain belonging to *chains* in the band

1 $info\_band \leftarrow$ InfoVirtualBand(*nodes*, Ch_j, Ch_k, Ch_i)
2 $l\_chains\_in\_band \leftarrow$ exist_chain_in_band_logic($info\_band$)
3 $exist\_chain \leftarrow$ len($l\_chains\_in\_band$) > 0
4 **return** $exist\_chain$

Generating new nodes

New nodes must be generated because the final chains cannot have holes. There are two situations where new nodes are generated. One case is when two chains are connected (Algorithm 8). In general, chains to be connected have an angular distance larger than two rays between the endpoints. A ray defines a node, which gives the direction and distance to the pith. We know the rays that lie between the endpoints that will be connected. Therefore, we linearly interpolate the position of the new nodes between the endpoints

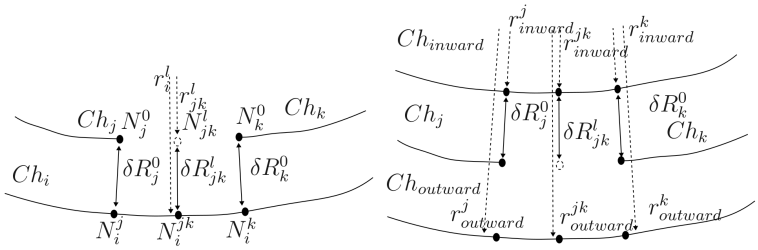


Figure 4.10: Nodes interpolation between chain endpoints a) One support chain b) Two support chains.

(for each ray direction, θ_i, the radial distance for the new nodes is generated). When a chain is completed, another similar situation is given (see Algorithm 9). In this case, node interpolation is made between chain endpoints. Figure 4.10a illustrates the situation described above. The angle of node N_{jk}^l is known, θ_i^l. The radial distance to node N_i^{jk} is r_i^l. We must compute the radial distance r_{jk}^l. As known variables we have: δR_j^0, which is the euclidean distances between nodes N_j^0 and N_i^j , and δR_k^0 , which is the distance between nodes N_k^0 and N_i^k. Additionally, the chain endpoint angles are also known θ_j^0 and θ_k^0, and the radii. Therefore r_{jk}^l is computed as

$$\begin{cases} \delta R_{jk}^l &= sign \times \frac{\delta R_j^0 - \delta R_k^0}{d_a(\theta_j^0, \theta_k^0)} \theta_i^l \\ r_{jk}^l &= \delta R_{jk}^l + r_i^l \end{cases} \tag{4.6}$$

Where *sign* indicates if nodes are generated inward (-) or outward (+) from the support chain, and d_a is the angular distance defined in Equation (4.4).

The other case occurs when node interpolation is performed between two support chains, as in the Postprocessing step (see Section 4.4.6). In this case r_{jk}^l is computed as (see Figure 4.10b)

$$\begin{cases} \delta R_{j\_ratio}^0 &= \frac{\delta R_j^0}{r_{outward}^j - r_{inward}^j} \\ \delta R_{k\_ratio}^0 &= \frac{\delta R_k^0}{r_{outward}^j - r_{inward}^j} \\ \delta R_{jk}^l &= \left(\frac{\delta R_{j\_ratio}^0 - \delta R_{k\_ratio}^0}{d_a(\theta_j^0, \theta_k^0)} \theta_i^l + \delta R_{k\_ratio}^0 \right) \times \left(r_{outward}^{jk} - r_{inward}^{jk} \right) \\ r_{jk}^l &= \delta R_{jk}^l + r_{inward}^{jk} \end{cases} \tag{4.7}$$

Finally, once the euclidean distance to the pith (r_{jk}^l) is determined for direction θ_i^l, node cardinal coordinates must be computed : $x_i \leftarrow x_{pith} + r_{jk}^l sin(\theta_i^l)$ and $y_i \leftarrow y_{pith} + r_{jk}^l cos(\theta_i^l)$.

4.4.6 Postprocessing

This final stage aims to complete the remaining chains, further relaxing the conditions. Many chains are completed at this stage ($size = Nr$). We call them *rings*. We still have some non-closed chains that can be noisy or be part of a ring but have not been completed for some reason. We use neighborhood chains to close or discard these remaining chains. Figure 4.11a illustrates a typical situation, with the closed chains in blue and the non-closed ones in red. The method iterates from the innermost to the outermost region. In this example, regions 0, 1, and 2 do not contain any candidate chains for connection. However, in region 3, intersecting chains, Ch_0, Ch_1, and Ch_2, are present. As shown in Figure 4.11b, chains are adjusted to eliminate intersections. Ch_2's endpoint intersects Ch_1 along ray r_m, leading to split Ch_1 into two subchains, with endpoints at rays r_{m-1} and r_{m+1}. A similar adjustment is applied to Ch_2, split along ray r_l. The next step consists of reconnecting chains if the connectivity goodness conditions are satisfied (Equation (4.5) with Table 4.1's last column parameters). Figure 4.11c illustrates the reconnection of chains.

Finally, in the last step of postprocessing, if the angular length of the chains within the region exceeds 180 degrees[2], as illustrated in Figure 4.11c, we consider that these incomplete chains contain sufficient information about the ring. The chains are completed in such cases, as shown in Figure 4.11d, by interpolating between the region's boundary rings in the positions of the existing chains (see Line 4).

Algorithm 11 describes the logic of the postprocessing stage (line 6 in Algorithm 5). In line 1, the regions are constructed. A region is defined by two complete chains (i.e., chains with a total number of nodes equal to Nr) and the set of incomplete chains between them. In line 2, the first region is extracted (the innermost one) to initiate the loop. The main loop, which iterates over the regions, spans lines 3 to 22.

In line 4, the inner ring (*ring*1), the outer ring (*ring*2), and the chains within the region (*chains_in_region*) are extracted. From lines 5 to 16, the algorithm iterates over the internal chains in *chains_in_region*, attempting to connect them as described at the beginning of the section. In line 7, all candidate chains (*candidates*) that intersect *chain* at its endpoints are identified. In line 8, these intersecting chains are split. Then, in line 9, the subset of non-intersecting chains within *chains_in_region* is selected. In line 10, the closest chain is chosen from both sets: *candidates* and *non_intersecting_chains*.

If the candidate pair satisfies the connectivity conditions, the chains are merged (line 12). Unlike in Algorithm 7, in this postprocessing stage, the non-overlapping-chain condition is not required (i.e., the term **not** *ExistChainOverlapping* in Equation (4.5) is not enforced). In line 13, the list *chains_in_region* is updated. In line 14, if *chain* has more than $0.9 \times Nr$ nodes (a fixed threshold), it is completed.

Once all internal chains in the current region have been processed, the loop terminates. Finally, any chain with an angular span exceeding 180 degrees is considered complete. For this purpose, chains are first sorted in descending order (line 17). Then, the longest chain with more than $0.5 \times Nr$ nodes is completed (see Algorithm 9). When a chain is completed, the for-loop between lines 18 and 21 is exited, and the region is split into two subregions (line 22). The inner subregion will be processed in the next iteration. If no

[2]This parameter is estimated based on the biological structure of the disc.

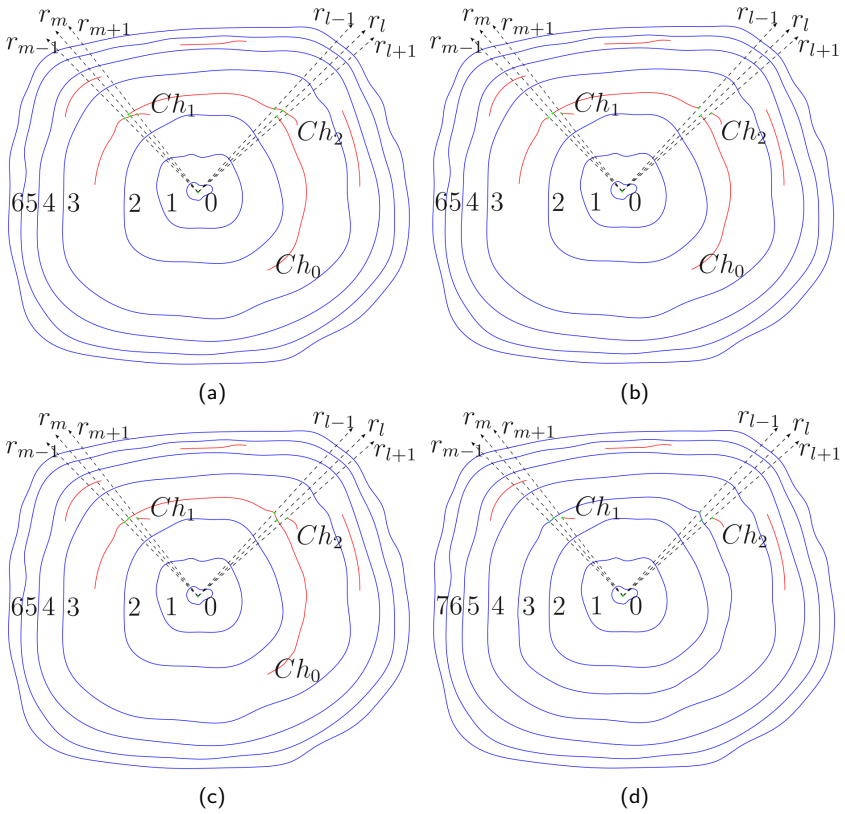


Figure 4.11: Postprocessing general logic. a) The input is a list of chains, with completed chains in blue and incomplete chains in red. Regions are labeled from 0 to 6. b) Chain Splitting. Chains Ch_1 and Ch_2 are split at the endpoints in rays r_m and r_l to avoid intersections. c) Chain Connection. Chains that satisfy the connectivity goodness condition are connected. d) Chain Completion and Region Addition. Chain Ch_0 is completed, adding a new region.

chain is completed, the algorithm proceeds to the next outer region relative to the current one.

4.4.7 Implementation details

The implementation was made in Python 3.11. The *README.md* file on the code repository contains all the information needed to run the code. The program requires, as input, an image of a tree slice and the pith position. Table 4.2 summarizes the method parameters, which the user can modify if needed. As output, the method returns a JSON file with the tree-ring positions in Labelme format [59]. Note that the values of parameters α, N_r, and m_c are fixed once and for all.

Algorithm 11: PostProcessing

```
   Input: chains
   Output: A list of post-processed chains
1  regions ← get_regions(chains)
2  current ← get_next_region()
3  while current do
4      ring1, chains_in_region, ring2 ← get_region_elements(current)
       /* Merge chains in region                                    */
5      chain ← get_next_chain(chains_in_region)
6      while chain do
7          candidates ← find_candidates_intersecting_chains(chain,
             chains_in_region)
8          candidates ← split_chains(chain, candidates)
9          non_intersecting_chains ←
             find_non_intersection(chains_in_region, chain)
10         Ch_k, E_j ← find_closest(non_intersecting_chains + candidates,
             chain)
11         if connectivity_goodness_condition( chain, Ch_k) then
12             chain ← merge_two_chains(chain, Ch_k, E_j, ring1, ring2) //
                 see Algorithm 8
13             chains_in_region ←
                 update_chains_list(chains_in_region,chain,Ch_k)
14         if chain_nodes(chain) ≥ 0.9 × Nr then
15             close_chain(chain, ring1, ring2) // see Algorithm 9
16         chain ← get_next_chain(chains_in_region)
       /* Close chains with an angle domain higher
          than 180 degrees                                         */
17     chains_in_region ← sort_chains(chains_in_region)
18     for chain in chains_in_region do
19         if chain_nodes(chain) ≥ 0.5 × Nr then
20             close_chain(chain, ring1, ring2) // see Algorithm 9
21             break
22     current ← get_next_region(current)
23 return chains
```

Table 4.2: CS-TRD method parameters.

	stage	Parameter	Description	Default
Basic	Edges detector	–sigma	Gaussian filtering σ	3
	Preprocessing	–height	Rezised image height	None
		–width	Rezised image width	None
	Filtering, sampling connect		Pith Position	Required
Advanced	Edges detector	–th_low	Gradient threshold low	5
		–th_high	Gradient threshold high	15
	Edges filtering	–alpha	Collinearity threshold (α)	30°
	Sampling	–nr	Number of rays (N_r)	360
		–min_chain_lenght	Minimum chain length (m_c)	2

4.5 DeepCS-TRD, a Deep Learning-based Cross-Section Tree Ring Detector

In this section, the edge detection module from the CS-TRD method is replaced with a U-Net [53] model, a deep learning architecture designed explicitly for semantic segmentation. U-Net architecture has demonstrated remarkable results in tree ring delineation of cores [10] and in microscopy images of shrubs [17].

By replacing the edge detection module in the CS-TRD method with this deep learning-based model, we successfully extended its application to a diverse range of species and growth forms. Notably, this enhanced approach works with angiosperm trees (i.e. *Gleditsia triacanthos*) and shrubs prepared using microscopy (i.e. *Salix glauca*). We refer to this new method as DeepCS-TRD.

4.5.1 Algorithm

The DeepCS-TRD method maintains the principal steps of the CS-TRD but with a substantial modification: replacing the Canny-based edge detection step with a module based on the U-Net architecture [53] trained to segment ring boundaries. Algorithm 12 shows the pseudo-code of this step, which substitutes lines 2 to 4 in Algorithm 5. The method's inputs are an RGB image of a cross-section (Im_{in}) and a parameter to indicate the image is split into tiles of (*tile_size*) size, following the overlap-tile strategy described in [53]. Additionally, the method allows to apply a Test-Time Augmentations (TTA) [28] strategy, which consists of *total_rotation* rotations around the pith location (c_y, c_x). The method produces a list of continuous ring edge curves (l_ch_s) and a list of pixels nodes (l_nodes_s) which are the input to the *spider web* model. In lines 1 and 2, the method initializes the TTA rotation domain angles Ω and the probability map P. Then, between lines 3 and 7, the inference loop iterates between the elements $\theta \in \Omega$. In line 4, Im_{in} is rotated an angle θ and stored in $I_{rotated}$. In line 5, the ring boundaries are generated with the *ring segmentation model* as illustrated in Figure 4.12. First, if needed, the image is split into tiles with an overlapping of 10% (to avoid the tile border effects). The tiles are zero-padded when necessary to ensure compatibility with the stride size (*tile_size*). Then, a probability map is computed for each tile with a U-Net network. Finally, the probability map for the whole disc ($P_{rotated}$) is built (with the original image size). The average between the tile's probability map is assigned to the overlapping area.

Algorithm 12: Deep Contour Detector

Input: Im_{in}: RGB slice image;
$tile\_size$: Tile size (0 if no patching is applied);
$total\_rotations$: Number of rotations applied to the image;
cy, cx: x and y coordinates of the pith position;
Output: l_ch_s: list of continuous curve edges;
l_nodes_s: list of nodes of the curves;

1 $\Omega \leftarrow$ arange$(0, 360, 360/total_rotations)$; // Define rotation angles
2 $P \leftarrow$ Zero matrix of the same size as Im_{in}; // Initialize the probability map
`// Process each rotation angle`
3 **for** $\theta \in \Omega$ **do**
4 $\quad I_{rotated} \leftarrow$ rotateImage$(Im_{in}, cy, cx, \theta)$; // Rotate the image around the pith position
5 $\quad P_{rotated} \leftarrow$ ringSegmentationModel$(I_{rotated}, tile_size)$; // Apply the segmentation model. See Figure 4.12.
6 $\quad P_{aux} \leftarrow$ rotateImage$(P_{rotated}, cy, cx, -\theta)$; // Undo the rotation of the probability map
7 $\quad P \leftarrow P + P_{aux}$; // Accumulate the rotated probability map
8 $P \leftarrow P/total\_rotations$; // Average the probability map
9 $M \leftarrow P \geq 0.2$; // Threshold the accumulated probability map
10 $S \leftarrow$ skeletonize(M); // Skeletonize the binary mask
11 $m\_ch_e \leftarrow$ findContoursCurves(S); // Extract ring curves from the skeleton
12 $Nx, Ny \leftarrow$ getNormalComponent(m_ch_e); // Compute normal directions for each edge
13 $m\_ch_f \leftarrow$ filterEdges(m_ch_e, Nx, Ny); // Filter edge curves. See Equation (4.8)
14 $l\_ch_s, l\_nodes_s \leftarrow$ samplingCurves(m_ch_f); // Convert curves to the Spider-Web format
15 **return** l_ch_s, l_nodes_s; // Return the results

Once the probability map of the whole image ($P_{rotated}$) has been obtained, the rotation of angle θ needs to be inverted (line 6, angle -θ). Then, in line 7, the current probability map (P_{aux}) is accumulated into P. In line 8, the average probability map is obtained. In line 9, the probability map P is binarized with a 0.2 threshold to generate the mask M. This low threshold is deliberately chosen, considering an additional filtering step in line 13 that further refines the results. In line 10, this mask is skeletonized to improve the tree-ring curve precision. Finally, curves are extracted from the skeleton S using the Suzuki method [57] implemented in the function *findCountour* of the OpenCV library [3]. Figure 4.13 illustrates the lines 8 to 11 steps. Figure 4.13b shows the probability map P for a disc region. Brighter pixels indicate a higher probability of belonging to a ring. Figure 4.13c shows the skeleton S overlaid on the mask M, with colors representing curve labels. The skeleton corresponds to the center of the mask. Finally, Figure 4.13d presents

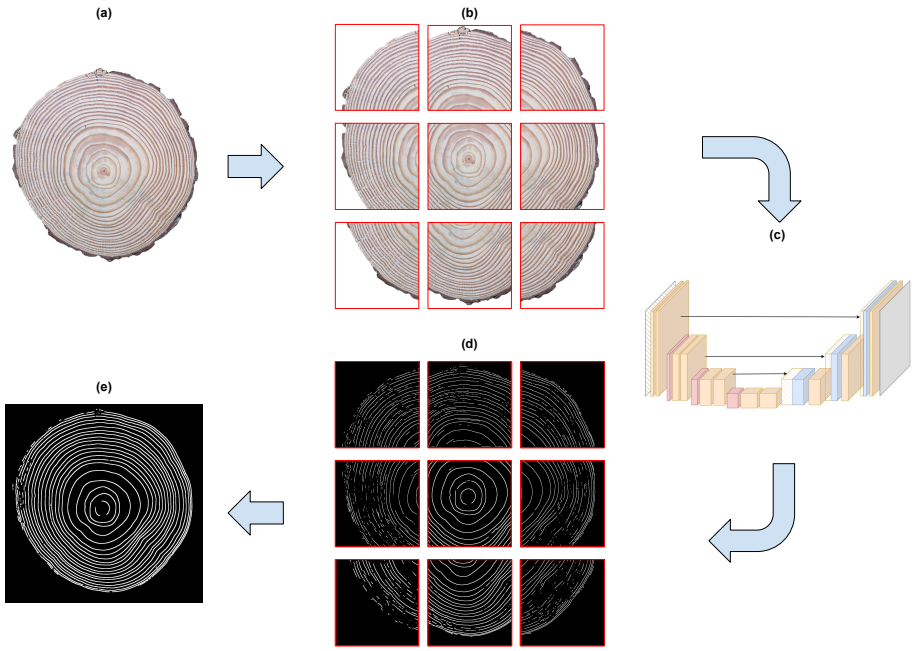


Figure 4.12: Ring Segmentation Model Inference. The input to the model is a complete cross-section disc (a). (b) the disc is divided into square tiles. (c) and (d) Each tile is processed individually using the U-NET network to generate its corresponding probability map. Finally, the tile predictions are combined to reconstruct the probability map for the entire sample (e). This procedure corresponds to line 5 of Algorithm 12.

the extracted curves overlaid on the original image.

The normal component of each pixel belonging to a curve in m_ch_e is computed using a straightforward procedure (line 12). The pixels of each curve are ordered sequentially from one end to the other. The tangent direction of pixel p_0 is $\vec{T_{p_0}} = p_1 - p_{-1}$, where p_{-1} and p_1 are the preceding and succeeding pixels along the curve, respectively. The normal direction $\vec{N_{p_0}}$ is then computed as a vector perpendicular to $\vec{T_{p_0}}$, given by $\vec{N_{p_0}} = \left(-T_{p_0,y}, T_{p_0,x}\right)$, where $\vec{T_{p_0}} = (T_{p_0,x}, T_{p_0,y})$. This ensures that $\vec{N_{p_0}}$ is orthogonal to the curve at p_0. The point c is the pith location (c_x, c_y). In line 13, the angle δ between $c\vec{p}_0$ and $\vec{N_{p_0}}$ is computed as $\delta\left(c\vec{p}_0, \vec{N_{p_0}}\right) = \arccos\left(\frac{c\vec{p}_0 \cdot \vec{N_{p_0}}}{\|c\vec{p}_0\| \|\vec{N_{p_0}}\|}\right)$

We filter out all *pixels* p_0 with normals not collinear and outbound oriented concerning the ray's direction at that point:

$$\alpha \leq \delta(c\vec{p}_0, \vec{N_{p_0}}) \leq 180 - \alpha \tag{4.8}$$

We set $\alpha = 45$ degrees. When a pixel p_0 is removed, the curve to which p_0 belongs is split at that position. This step is straightforward because m_ch_e is a matrix of dimensions $N \times 2$. Each curve is separated from the others by a row with values $(-1, -1)$. Therefore, if a pixel does not satisfy the condition in Equation (4.8), we insert the vector $(-1, -1)$ at its position.

Finally, in line 14, the curves in m_ch_f are converted into the format required by the

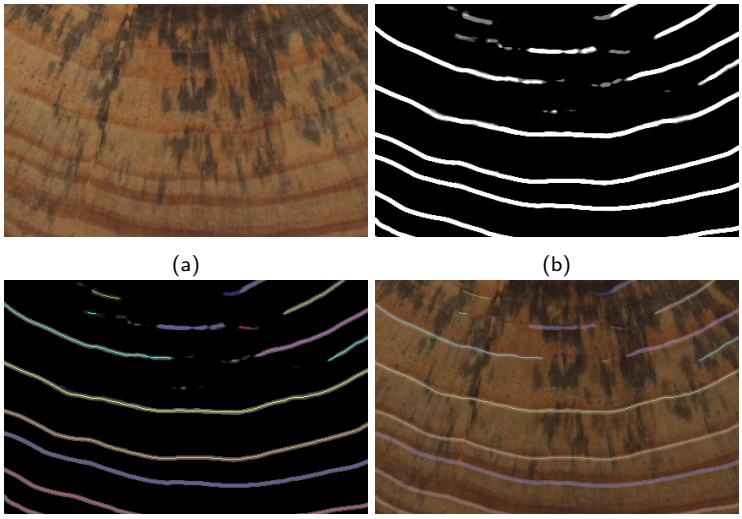


Figure 4.13: Stages for extracting the ring edge curves. (a) Zoomed-in view of the RGB image. (b) The probability map of the image section is shown in (a). (c) Probability map with overlaid edge curves. (d) Extracted edge curves overlaid on the image.

spider web model, structured around a central *pith*, the origin of the *rays*. Each *curve* is represented as a set of points, with *nodes* in the intersections of the curve with the *rays*. The resulting sampled *curve* is called a *chain*. These *chains* have the notable property of being non-intersecting, inspired by the biological characteristics of tree rings observed in conifer species. More details can be found in Section 4.4.

Preprocessing The background was removed from all images using the U^2-Net network [50] and manually corrected when necessary. Furthermore, portions of the image background were cropped to focus on the cross-section so that the minimum distance between the disc and the image edges is 50 pixels. Images from all datasets were resized to 1504x1504 pixels using Lanczos interpolation [16]. To complete the image size, if the smallest dimension of the resized image does not reach the new dimension, a 255-value padding was applied, maintaining the aspect ratio.

4.5.2 Network Training

Non-overlapping square tiles were used for training the U-Net network. Only the patches with ring presence of each image sample were used. The tile size is a hyperparameter of the training step (see Appendix B.2) where 60% of the samples were used for training, 20% for validation and 20% for assessing the full tree-ring detection pipeline. Training and validation subsets were split into patches to train and select the network's hyperparameters. We also trained the network to predict the ring boundaries on the complete disc sample.

The network was trained over 100 epochs where the Adam optimizer [29] was used with an initial learning rate of 1e-3, while a cosine annealing schedule [37] was applied

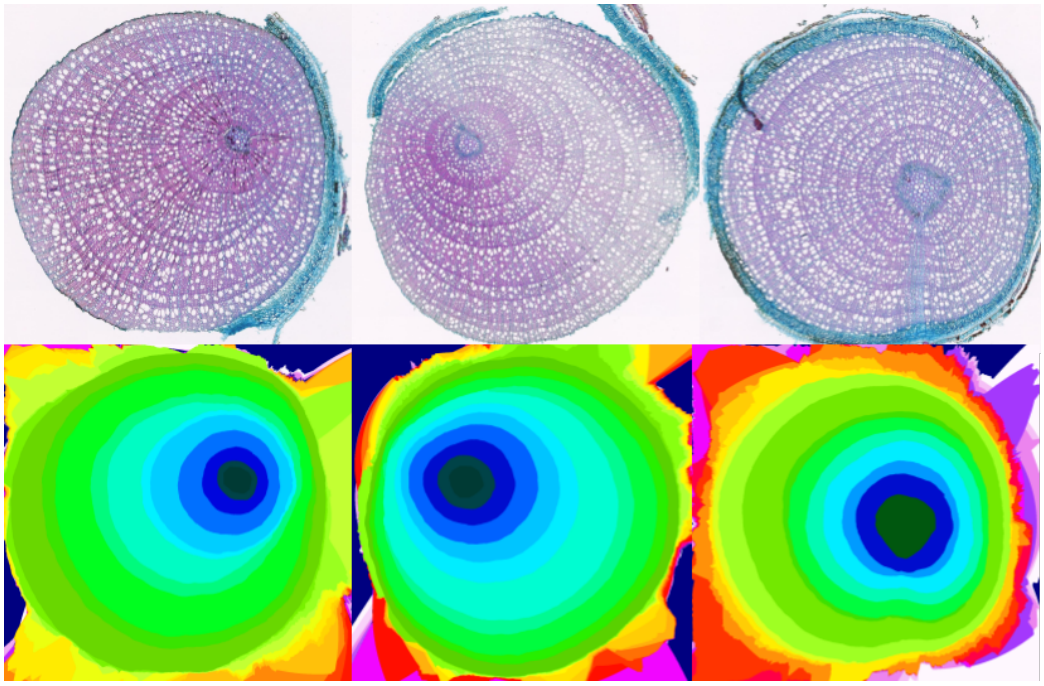


Figure 4.14: Samples from the INBD *EH* train set. In the first row are the original samples, and in the second row are the inference results of the corresponding samples.

to adjust the learning rate dynamically during training. The Dice Loss function was employed to optimize the model's performance. The Resnet18 backbone, pre-trained on ImageNet, was also used to enhance feature extraction. The network's weights that result in the lowest validation loss during training are selected as the optimal model.

Regarding data augmentation, in 50% of the dataset images, randomly selected, we generate three augmented images: one with a random rotation around the pith, another with occlusions simulating cracks or fungi, and a third with elastic deformations.

4.6 Iterative Next Boundary Detection for Instance Segmentation of Tree Rings

In this section, we describe the Iterative Next Boundary Detection for Instance Segmentation (INBD), a deep neural network-based approach originally presented in [17] and developed for automatically delineating annual rings in microscope images of shrubs. Figure 4.14 illustrates some examples of those images, part of the *EH* dataset [17]. We train the network to automatically delineate tree rings in cross-section images of *Pinus taeda*, *Gleditsia triacanthos*, and *Salix glauca*. As seen in Figure 4.18, these images have varying resolutions, and their textures differ significantly due to the species' characteristics.

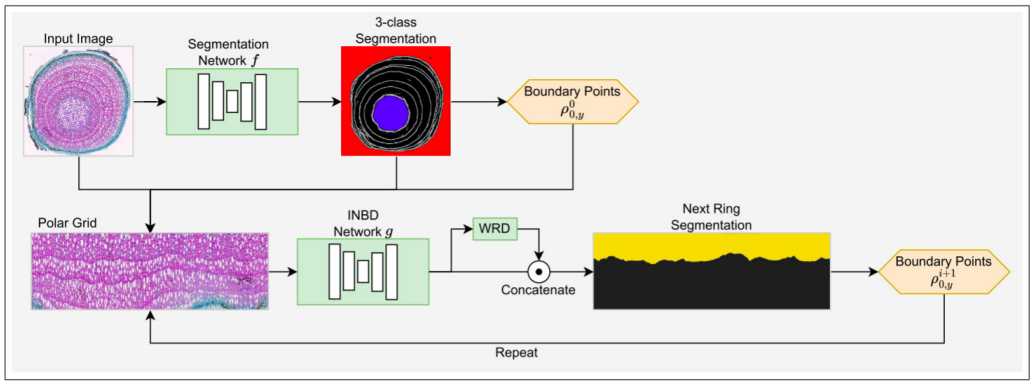


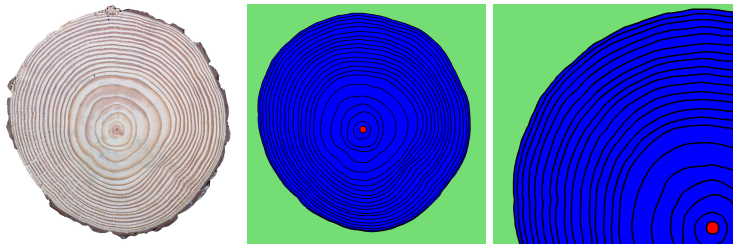
Figure 4.15: Overview of the INBD pipeline. An input image is first passed through a U-Net network that detects three classes: background (red), ring boundaries (white), and the pith region (blue). A polar grid is sampled starting from the border of the detected pith region (the first ring) and passed to the main INBD network that detects the next ring. This process is repeated until the background is encountered. Image taken from [17].

4.6.1 Algorithm

Figure 4.15 illustrates the INBD pipeline. The input of the pipeline is a cross-section of a tree image. The input image is first segmented into the background, ring boundaries, and pith region. The cross-section image is then transformed into polar coordinates, with the pith's center as the origin. Rectangular image patches are extracted iteratively, allowing individual ring segmentation from the inner one (closer to the pith) to the outer rings (closer to the bark). In the second step of the INBD method, the segmented regions obtained in the first step are used as follows: the pith's center is used as the origin of the polar coordinates transformation, the ring boundaries are used to determine the width of the rectangular image patches and the background is used as a stop criterion in the iterative process. The second step can be interpreted as a refinement of the pixels assigned to the ring boundary category in the first step, followed by a transformation of these pixels into the mathematical object curve. Both stages utilize a U-Net network. The following paragraph provides a detailed description of each step.

Segmentation Network f This is a semantic segmentation network trained to detect three classes: background, ring boundaries, and the center ring (pith). The network and its output is denoted as $f(\mathbf{I}_i) = (y^{bg}_{pred\_i}, y^{bd}_{pred\_i}, y^{ct}_{pred\_i})$ when applied on image $\mathbf{I}_i$ where $y^{bg}_{pred\_i}, y^{bd}_{pred\_i}, y^{ct}_{pred\_i}$ are the output of the classification layer of the same height and width than $\mathbf{I}_i$ of the classes background (bd), ring boundary (bg) and center (ct) respectively[3]. Concerning the network architecture, a U-Net network is used with mobilenet3l backbone pretrained on ImageNet, trained to optimize a combination of Dice and Binary Cross Entropy losses:

[3]Note that the network outputs are not probabilities. To convert them into probabilities, the sigmoid function is applied, as seen in loss functions like Binary Cross-Entropy or Dice Loss.



(a) (b) (c)

Figure 4.16: INBD training data (a) F02c *Pinus taeda* image sample (b) Labels for the tree classes: background in green, rings boundary in black, and center pith in red (c) Zoom-In of (b). Ring boundary thickness is set to 3 pixels. Additionally, ring regions are colored in blue, though they are not segmented.

$$L_f = \frac{1}{N}\sum_{i=0}^{N-1}\Big(\frac{BCE(y^{bg}_{pred\_i}, y^{bg}_{tg\_i})}{100} + Dice(y^{bd}_{pred\_i}, y^{bd}_{tg\_i}) + \frac{BCE(y^{ct}_{pred\_i}, y^{ct}_{tg\_i})}{10}\Big)$$

Where $y^{bg}_{tg\_i}$, $y^{bd}_{tg\_i}$, $y^{ct}_{tg\_i}$ are binary masks of the same height and width than $\mathbf{I}_i$ of the classes background (bg), ring boundary (bd) and center (ct) respectively and N is the size of the training set. The different weights assigned to each loss component $(\frac{1}{100}, 1, \frac{1}{10})$ seek to compensate the class imbalances. Figure 4.16 illustrates a sample image of *Pinus taeda* along with its three-class annotations. In Figure 4.16b, the background class (y^{bg}_{tg}) is represented in green, the boundary class (y^{bd}_{tg}) is shown in black, and the center class (y^{ct}_{tg}), the pith, is highlighted in red. The ring regions are colored in blue, though they are not segmented.

During inference, to identify the pixels belonging to each class, each matrix $(y^{bg}_{pred_i}, y^{bd}_{pred_i}, y^{ct}_{pred_i})$ is thresholded[4]. Values greater than 0 are set to 1, and the remaining values are set to 0.

Polar Grid Starting from the boundary center of the pith which is the contour of the region masked by y^{ct}_{pred}, iteratively a polar grid (hereon P^i) is generated which is the input to the *INBD Network* g. The black dots in Figure 4.17 illustrate the grid P^i generated at iteration i over the image sample. The polar grid *Origin*, illustrated in the figure, is computed as the average of the pith boundary points. Polar grid dimensions are $P^i \in \mathcal{R}^{N\times M}$ with N=256 a fixed resolution in the radial axis and M an adaptive resolution in the angular axis.

Sampling points (black dots in Figure 4.17) are expressed in polar coordinates as $(\rho^i_{xy}, \varphi^i_{xy})$, with $x \in [0, N]$ and $y \in [0, M]$ indices within the grid. Therefore, the boundary point radii for the starting boundary (the pith contour) are $\rho^0_{0,y}$. The length of the grid in the radial dimension in pixels (i.e. ρ_{Ny}) is computed as the distance to the closest positive value (1) in y^{bd} for each angle φ. The length is set to $1.5 \times 95\%$-th percentile of these

[4]Note that the network classification output can takes values below 0.

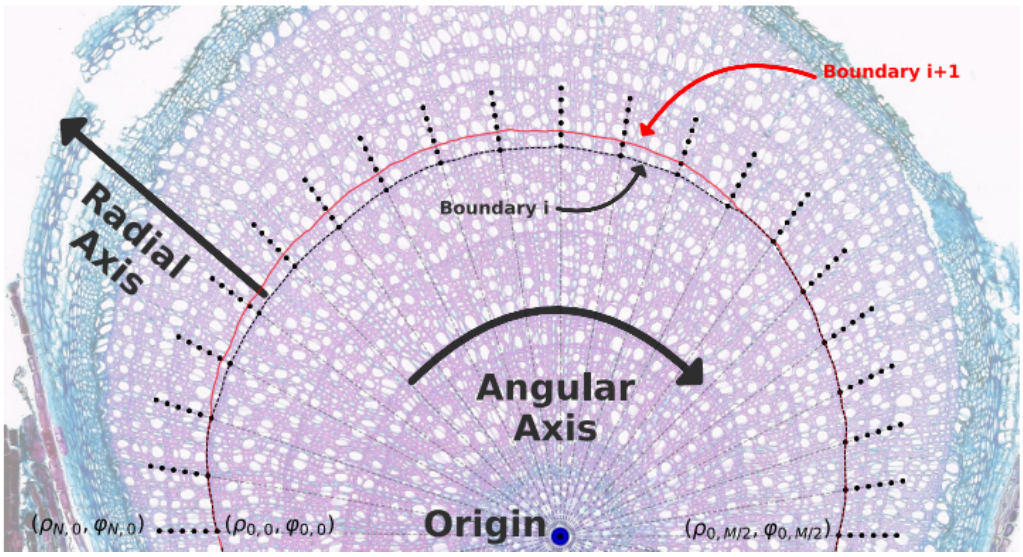


Figure 4.17: Polar grid sampling. The predicted boundary by network f is $Boundary_{i+1}$. $Boundary_i$'s elements determine the grid starting points. Image taken from [17].

distances (a fixed value), to make sure that most points are included and, at the same time, filter out the outliers. The remaining radial values ρ are uniformly distributed along the range: $\rho_{xy} = \frac{1}{N}(p_{Ny} - \rho_{0y})x + \rho_{0y}$.

The angular resolution M is variable between grids, in such a way that the angles φ have an approximately uniform Euclidean distance to each other across rings: since the outer rings have a larger circumference than the inner ones, they should be sampled at a higher angular resolution M. The value M^i for P^i is computed from the previous starting boundary average radii:

$$M^i = \alpha \frac{1}{M^{i-1}} \sum_{y=0}^{M^{i-1}} \rho_{0,y}^{i-1}$$

with $\alpha = 2\pi$, a hyperparameter that controls the general density of M^i. The angles φ are spaced uniformly along the full circle: $\varphi_{xy} = \frac{1}{M} 2\pi y$.

INBD Network g The main network, denoted as g, is another 2D convolutional segmentation network that classifies each pixel as belonging to the next ring or not. This second network has a similar architecture as the first one, except that it replaces standard 2D convolutions with circular convolutions to account for the whole circular structure. Unlike network f, which classifies three classes, this network classifies only one. The input is the tensor $I^i \in \mathcal{R}^{C \times N \times M}$ where the channel dimension C=6 is composed by the RGB channels of the input image, the detected 'background' and the 'boundaries' outputs y^{bg} and y^{bd} from network f, and the normalized radii ρ concatenated together. The pipeline allows adding an extra channel related to the output of the wedging ring detection module (WRD block in Figure 4.15). However, this feature is not used in this thesis, as the

species analyzed do not present wedging rings. The INBD code includes a flag to enable or disable the WRD block accordingly.

The main loss L_{CE}^{cls} for the network g is the standard cross-entropy to classify each pixel in the polar grid as belonging to the next ring or not, according to the annotation A^i, sampled on the polar grid.

To perform inference of the next ring's boundary points $\rho_{0,y}^{i+1}$, they select the last positive point in the output $g(I^i)$ column-wise, where it is unambiguous:

$$X_y = \{x \mid g(I^i)_{x,y} = 1\}$$

$$\rho_{0,y}^{i+1} = \begin{cases} \rho_{\max X_y, y}^{i}, & \text{if } \max X_y = \min \bar{X}_y - 1 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Ambiguous values are linearly interpolated in polar coordinates. This detection process is repeated iteratively with the new predicted ring boundary points i+1 as the starting point to detect the ring i+2 until the background y^{bg} that was detected by the segmentation network f is reached.

4.6.2 Network Training

Both networks (f and g) must be trained. Since boundary predictions are generated iteratively, errors caused by the earlier rings propagate onto the later rings. The authors proposed an *iterative training* procedure for INBD network g to mitigate the ring propagation errors. The Listing 4.1 shows the pseudo-code for one training epoch, incorporating previous (possibly faulty) predictions as the starting point for the polar grid. The number of iterations per epoch (n = 3) is a hyperparameter of the training procedure.

Listing 4.1: Pseudo-code for one training epoch

```
for (image I, annotation A, ring i) in dataset:
    L_g = 0
    ρ^i_{0,y} = boundary_from_annotation(A, i)
    loop i = i..i + n:
        ρ̃^i_{0,y} = augment(ρ^i_{0,y})
        I^i = sample_polar_grid(I, (ρ̃^i, φ̃^i))
        y^i = g(I^i)
        L_g += compute_loss(y^i, A^i)
        ρ^{i+1}_{0,y} = compute_boundary(y^i)
    backpropagate(L_g/n)
```

Besides data augmentation techniques such as pixel-wise color jitter operations, the authors employ specific polar grid augmentations, such as varying the boundary points. Both networks are trained sequentially with the Adam optimizer [29] for 100 epochs, an initial learning rate of 1e-3, and a cosine annealing learning rate schedule [37].

(a) UruDendro1 (b) UruDendro2 (c) UruDendro3a (d) DiskoIsland

Figure 4.18: Examples of the used datasets. Species are (a-b) *Pinus taeda*, (c) *Gleditsia triacanthos*, (d) *Salix glauca*. Acquisition conditions: (a-c) in a laboratory with a smartphone camera, sanded and polished. (d) stained with 1 % safranin and 0.5 % astrablue and permanently fixed with Eukitt (BiOptica, Milan)

Table 4.3: Dataset description.

Collection	**Samples**	**Tree-rings**	**Specie**	**Acquisition**
UruDendro1	64	1221	*Pinus taeda*	Smartphone
UruDendro2	53	1151	*Pinus taeda*	Smartphone
UruDendro3a	9	216	*Gleditsia triacanthos*	Professional Camera
DiskoIsland	50	654	*Salix glauca*	Scanner

4.7 Datasets

The datasets used in this study encompass various species as shown in Figure 4.18. Some with clear ring contrasts, such as the conifer *Pinus taeda*, and more challenging ones like *Gleditsia triacanthos* and *Salix glauca*. The acquisition methods vary: some images were captured using smartphone cameras, and others were obtained via a scanner.

Ring boundaries were annotated using the Labelme Tool [59] as described in Chapter 2. Additionally, a binary image mask was generated from the ring boundary annotations, with a thickness of 3 pixels.

Table 4.3 illustrates the datasets, including the number of samples and rings in each, the species, and the acquisition method.

4.8 Results and discussion

4.8.1 Metrics

We adopt the same evaluation metrics used by the authors of the INBD method: the mean Average Recall (mAR) as defined in [23], and the Adapted Rand error (ARAND) as defined in [1].

Mean Average Recall: Average Recall (AR) is computed for different Intersection-over-Union (IoU) thresholds between the ground truth ring and the detected ring [5]. The mean Average Recall (mAR) is then defined as:

$$\text{mAR} = \frac{1}{|T|}\sum_{t\in T}\text{AR}(t), \quad \text{with} \quad \text{AR}(t) = \frac{TP(t)}{TP(t)+FN(t)} \tag{4.9}$$

where $TP(t)$ and $FN(t)$ refer to the number of true positives and false negatives at IoU threshold t, respectively. The set of thresholds T ranges from 0.5 to 1.0 with a step size of 0.05, i.e., $T = \{0.50, 0.55, \ldots, 1.0\}$.

False positives are preferred to false negatives because while we can delete an incorrect ring with a click, tracing a new one is more time-consuming. In this sense, the mAR metric is particularly suitable for this application.

Adapted Rand Error: Let p_{ij} denote the probability that a randomly chosen pixel belong to a j-th ground truth ring and the i-th predicted ring. Define:

$$s_i = \sum_j p_{ij}, \quad t_j = \sum_i p_{ij}$$

Where s_i represents the probability that a randomly chosen pixel belongs to the i-th predicted ring. On the other hand, t_j represents the probability that a randomly chosen pixel belong to the j-th ground truth ring.

$$\text{sum}_p = \sum_{i,j} p_{ij}^2, \quad \text{sum}_s = \sum_i s_i^2, \quad \text{sum}_t = \sum_j t_j^2$$

sum_p represent the probability that two randomly chosen pixels belong to the same i-th predicted ring and j-th ground truth ring.

Then, precision and recall are computed as:

$$\text{Precision} = \frac{\text{sum}_p}{\text{sum}_t}, \quad \text{Recall} = \frac{\text{sum}_p}{\text{sum}_s}$$

The F-score and ARAND are defined as:

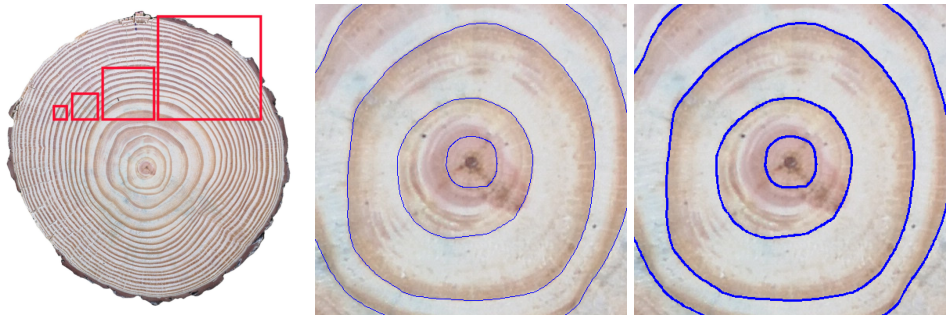
$$\text{F-score} = \frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}, \quad \text{ARAND} = 1 - \text{F-score}$$

The implementation provided by *scikit − image* is used `skimage.metrics.adapted_rand_error`.

4.8.2 Experiments

The CS-TRD method heavily relies on the edge detector stage. We tested different σ values for the Canny Devernay edge detector to get the one that maximizes the performance. After experiments, the σ value was fixed at 3 (see Appendix B.2.3).

[5] In this metric, a ring in the region delimited between two consecutive and concentric ring boundaries



(a) (b) (c)

Figure 4.19: Training parameters. (a) Tile sizes: 64, 128, 256, and 512 pixels. (b) Annual ring thickness of 1 pixel and (c) Annual ring thickness of 3 pixels.

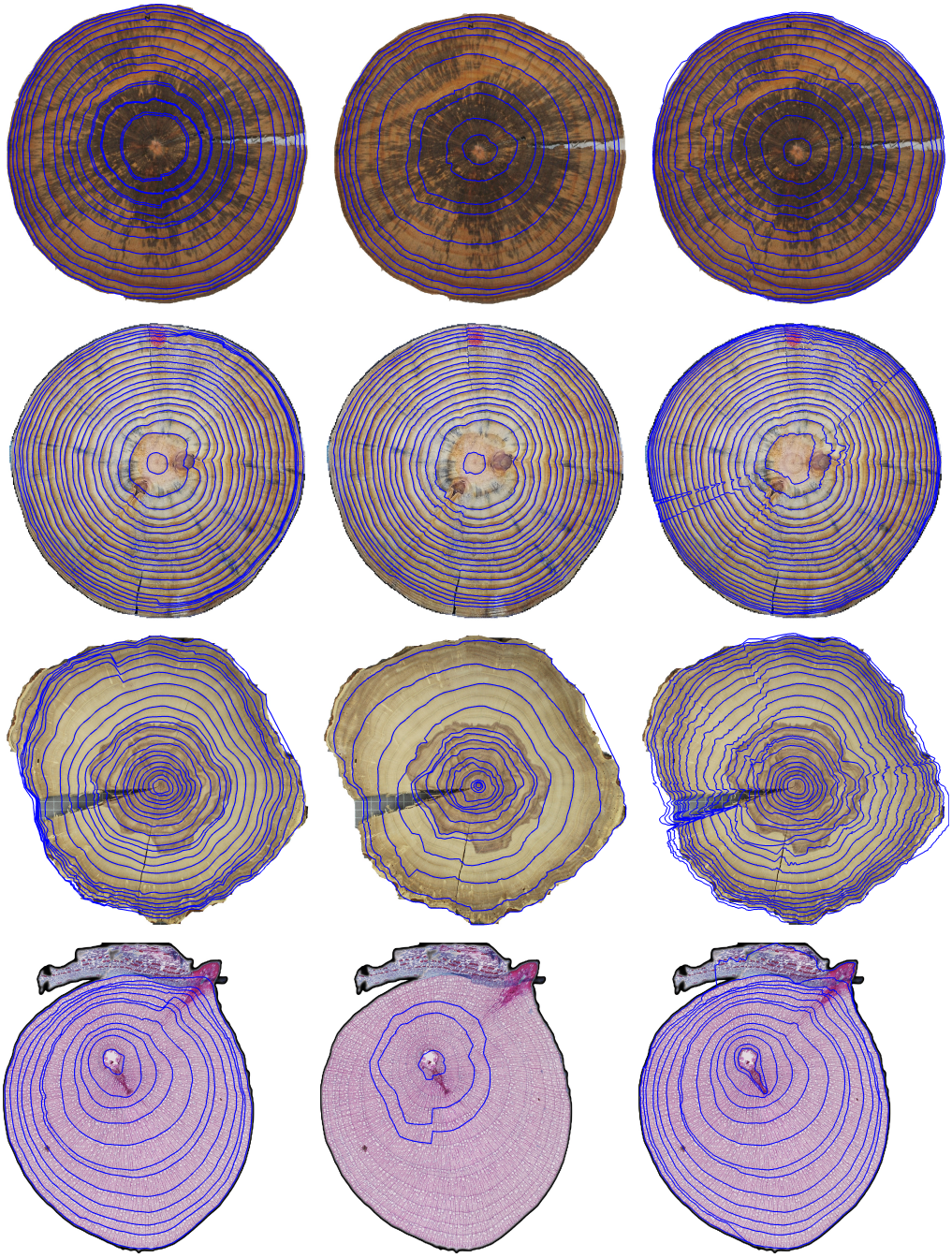
Table 4.4: Results on the test set for the four used datasets and the available SOTA methods: INBD [17] and CS-TRD as well as the proposed DeepCS-TRD. The best performance is in boldface. Uru1 stands for UruDendro1, and the same applies to UruDendro2 and UruDendro3 datasets. Disko stands for DiskIsland dataset.

	mAR ↑				**ARAND ↓**			
Method	**Uru1**	**Uru2**	**Uru3a**	**Disko**	**Uru1**	**Uru2**	**Uru3a**	**Disko**
CS-TRD	.787	.710	.007	.026	.093	.144	.466	.634
INBD	.846	.742	.200	**.735**	.081	.132	.494	**.099**
DeepCS-TRD	**.906**	**.832**	**.640**	.616	**.051**	**.089**	**.116**	.116

The DeepCS-TRD method involved two key parameters: *tile_size* and *total_rotations* (see Algorithm 12). We tested tile sizes of 64, 128, 256, 512, and 1504 (full resolution, no tiles) pixels for all the datasets. Figure 4.19a illustrates the tile sizes. For the *total_rotations* parameter, combinations of 0, 3, and 5 rotations were evaluated. The selection of this parameter was made by optimizing the validation subset. After the experiments, the parameter values were fixed at *tile_size* = 256 and *total_rotations* = 5 for the four datasets.

We noted that the thickness of the annotated ring boundary masks during training impacts the method's performance. The best value was 3-pixel in all datasets (see Figure 4.19b and Figure 4.19c). In consequence, DeepCS-TRD and INBD methods are trained using ring boundary thicknesses of three pixels. The two networks in the INBD method were trained following the author's procedure for 100 epochs, with the downsampling parameter set to 1, which yielded the best results.

For training the DeepCS-TRD and INBD method, clusterUy [43]'s infrastructure was used, which in our experiments consisted of an NVIDIA P100 GPU with 12 GB of VRAM and 40 GB of system RAM. Inference was performed on an Intel Core i5-10300H workstation with 16 GB of RAM and an NVIDIA GeForce GTX 1650 with 4 GB of VRAM.



(a) DeepCS-TRD (b) CS-TRD (c) INBD

Figure 4.20: Tree-Ring Delineation Results. Each column displays the ring boundaries produced by each method, shown in blue. Each row corresponds to a different disc sample. Column (a) DeepCS-TRD; Column (b) CS-TRD; Column (c) INBD. Note the presence of knots, cracks, fungus, and the differences between species. Note that the darker region in the third row does not correspond to the tree rings of that species.

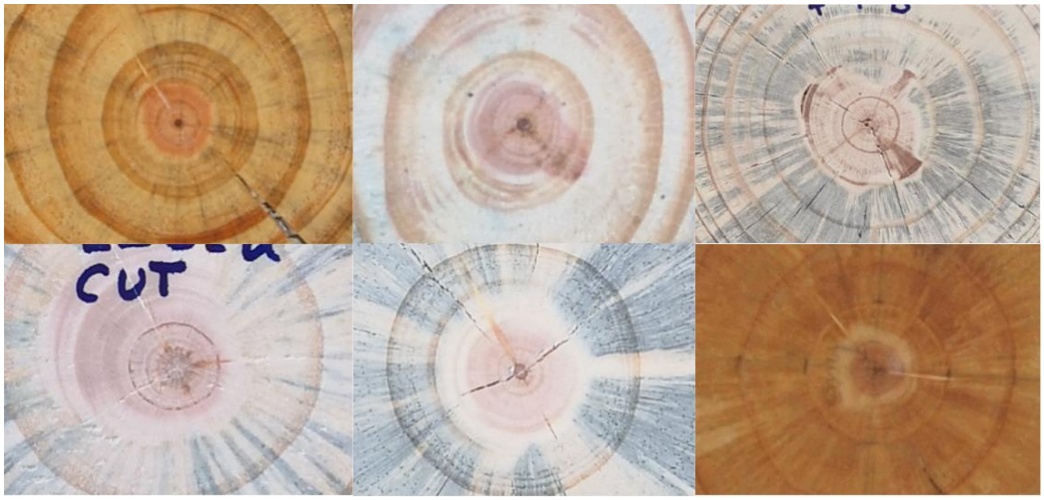


Figure 4.21: Pith samples from the UruDendro1 dataset. In the *Pinus taeda*, the characteristics of the pith present differences within the species.

4.8.3 Results

This section evaluates the performance of INBD, CS-TRD, and DeepCS-TRD on the test set across all datasets. The INBD method was modified to accept the pith boundary ground truth as input to ensure a fair comparison, as both DeepCS-TRD and CS-TRD take the pith locations as input but model them as a pixel [41]. Furthermore, in the *Pinus taeda* and *Gleditsia triacanthos* species, the pith detector network in the INBD method yielded unsatisfactory results. As shown in Figure 4.21, the pith characteristics vary significantly within the species. Additionally, the pith size, relative to the sample size, is much smaller in *Pinus taeda* than in Shrub introduced by the INBD authors (EH dataset specifically). As seen in Figure 4.14, the pith size for this species is much bigger than for the samples in the UruDendro dataset.

Table 4.4 presents the mAR and ARAND values for each method across all datasets. The proposed DeepCS-TRD method has the best results on all the UruDendro datasets. It achieves remarkable results with near-perfect ring detection from the pith to the bark in UruDendro1 and UruDendro2, as illustrated in Figure 4.20. Despite the INBD method yielding very good performance in most samples, in some situations, it generates notable propagation errors in the presence of significant fungal growth, cracks, or knots, due to the method's inherent limitations. When an error occurs, it propagates outward (see rows 1 and 2). In the sample of row 1, DeepCS-TRD detects some false rings near the center, but its remaining detections are highly accurate. In contrast, CS-TRD yields good results, but the rings are inaccurate due to the strong presence of fungus. In the sample in row 2, both DeepCS-TRD and CS-TRD performed well.

In the UruDendro3a dataset, DeepCS-TRD performs correctly; in the sample shown in Figure 4.20, row 3, it correctly delineates 15 out of 19 rings. In contrast, the INBD method detects only six rings. CS-TRD detects five rings but, interestingly, identifies the three innermost rings missed by the other methods (see the region around the pith in row 3 for all methods).

Table 4.5: Training time (in seconds) for the INBD and DeepCS-TRD methods on each dataset. The training time for each INBD network is reported separately.

Dataset	INBD			DeepCS-TRD
	Segmentation Network f	INBD Network g	Total	
Uru1	4492	84120	88612	13624
Uru2	12778	70203	82981	11504
Uru3a	5019	12715	17734	2377
Disko	2764	34101	36865	9839

Table 4.6: Average inference time (in seconds) for the INBD, CS-TRD, and DeepCS-TRD methods on each dataset. Datasets where a method does not produce accurate results are excluded (denoted by '-').

Dataset	INBD	DeepCS-TRD	CS-TRD
Uru1	18.6	22.5	25.6
Uru2	21.9	29.5	36.3
Uru3a	-	62.6	-
Disko	12.6	21.2	-

In the DiskoIsland dataset, DeepCS-TRD performs worse than the INBD method on average. In the sample in row 4, the INBD method correctly detects all the rings, while DeepCS-TRD misses the second ring. CS-TRD performs poorly in this dataset.

There are notable differences in training time between the DeepCS-TRD and INBD methods. The maximum training time for both methods was observed on the UruDendro1 dataset, requiring 25 hours for INBD and 4 hours for DeepCS-TRD, using the same hardware (see Table 4.5).

On the other hand, Table 4.6 presents the average inference time (in seconds) for each dataset and method. In the UruDendro1 and UruDendro2 datasets, INBD achieves the lowest inference time, at 18.6 and 21.9 seconds, respectively, followed by DeepCS-TRD, which takes 22.5 and 29.5 seconds. The CS-TRD method has the highest execution time, at 25.6 and 36.3 seconds for each dataset.

In the UruDendro3a dataset, the only method that produces reasonably accurate results is DeepCS-TRD, with an inference time of 62.6 seconds. Finally, in the DiskoIsland dataset, the INBD method has an inference time of 12.6 seconds, while DeepCS-TRD requires 21.2 seconds. Additionally, Appendix B.2.4 presents the performance of DeepCS-TRD without using tiles during training and inference for all the datasets.

4.9 Conclusions and future work

A new method for automatically delineating tree rings in conifer wood cross-section images, referred to as CS-TRD, is proposed. It is based on a classical Canny-Devarnay edge-detection algorithm to extract ring edge information. After edge detection, a post-processing logic inspired by the biological properties of tree rings is applied to transform them into curves, allowing the computation of ring properties such as area and perimeter.

We employ a model we refer to as the *spider web* model for this approach.

This algorithm achieves a performance of 0.787 in mAR and 0.093 in ARAND metrics on the UruDendro1 dataset. On the other hand, the UruDendro2 dataset achieves 0.710 and 0.144 in mAR and ARAND metrics, respectively. These results demonstrate accurate ring delineation, accelerating the precise computation of tree-ring properties.

An adaptation of the CS-TRD method is proposed, where the edge detection step is replaced with a U-Net deep convolutional network. We named this adaptation DeepCS-TRD. This modification allows us to apply the *spider web* model to another species, such as *Gleditsia triachantos* or *Salix glauca*, with different annual ring patterns than for the coniferous species. Additionally, the proposed method increases the performance of CS-TRD also for *Pinus taeda* as can be seen in Table 4.4; in this case, the accuracy (mAR) has improved from 0.787 to 0.906 in UruDendro1 and from 0.710 to 0.832 in UruDendro2. The downside is that this adaptation requires annotated images to train the network.

Additionally, the INBD method for tree-ring delineation in microscopy cross-section images of shrubs was assessed to delineate tree rings in conifer images taken with a smartphone, a different application domain. This method is based on the U-Net deep learning architecture and, like DeepCS-TRD, requires annotated data for training. It outperforms the CS-TRD method in all the datasets, particularly in UruDendro1 and UruDendro2, achieving an accuracy (mAR) of 0.846 and 0.742, respectively. In the UruDendro datasets, the INBD pith detector did not achieve satisfactory results.

The DeepCS-TRD method got a reasonable accuracy in the UruDendro3a dataset, scoring 0.640 in the mAR metric. Additionally, it has produced satisfactory results on the *Salix glauca* dataset, with an mAR of 0.616. The INBD method scored 0.735 (mAR).

Regarding the training step required by the DeepCS-TRD and INBD methods, DeepCS-TRD needed 4 hours, while INBD required 25 hours in the UruDendro 1 dataset. During inference, INBD required 18.6 seconds and DeepCS-TRD 22.5 seconds in the UruDendro1 dataset, while CS-TRD took 25.6 seconds. Execution time performance is similar in the UruDendro2 dataset; however, in the DiskoIsland dataset, DeepCS-TRD takes twice as long, requiring 21.2 seconds compared to the 12.6 seconds that INBD needed.

Further work can be done by acquiring and annotating more *Gleditsia triacanthos* image samples, which is essential to improve machine learning based automatic ring delineation methods.

Chapter 5

TRAS, an Interactive Software for Tracing Tree Ring Cross Sections

This chapter is based mainly on the work submitted to the Dendrochronology Journal special issue Trace 2024 entitled "TRAS, an Interactive Software for Tracing Tree Ring Cross Sections".

5.1 Introduction

In dendrometry studies, ring width is one of the primary parameters of interest, and it is common practice to take measurements from core samples. However, in tree trunks with high pith eccentricity, ring area often correlates better with climate conditions such as precipitation or temperature [34].

Storing samples in digital format (as images) facilitates sharing them with the community and validating results. The methods for digitizing tree-ring measurements are still predominantly manual. One significant challenge is the difficulty of digitizing disc or cores. High-resolution scanners (with a resolution of up to 3.9 microns per pixel) are available; however, they require samples to be completely flat, a condition that is difficult to achieve with cores or disc. Digital photography has emerged as a viable alternative for tree ring studies [13]. For example, [27] compares measurements taken from scanned and smartphone images, obtaining comparable results. However, proper camera calibration is essential when using images acquired with a camera.

Ring curves can be manually delineated from images using tools with graphical user interfaces, such as ImageJ or Adobe Illustrator [5], which allow users to measure areas defined by their input. Moreover, in [51], a system called Wood Image Analysis and Dataset (WIAD) is presented. It is an R package with a graphical user interface that enables manual ring-width measurement along a linear path and archival functionalities to store the image, the ring-width series, and metadata. However, manually delineating the complete rings in the stem is a time-consuming process.

Constantz et al. [5] proposed a Python package for studying growth ring patterns using 2D information in species with asymmetric growth characteristics, such as Hawaiian Sandalwood (*Santalum paniculatum*). They developed a free Python package that, based

on manual tree-ring annotations in an image, automatically computes multiple transects. However, a limitation of their approach is the lack of an automatic or semi-automatic method for ring delineation, making it more time-consuming than measuring ring width on core samples.

Some graphical software tools implement automatic ring detection based on edge information in cores rather than cross-section images [27, 42, 55]. These tools allow users to correct detection errors and perform different measurements.

Recent developments in core sample image analysis include the application of deep learning techniques, where semantic segmentation networks aim to mark the pixels that belong to the tree rings. In [10], a recall of 96% and a precision of 97% on scanned images of ring-porous wood core samples using a U-Net architecture [53] is reported, but the code is not made available. The authors of [47] trained a Mask R-CNN network over microscopy images of Norway Spruce (*Picea abies L.*), reporting a recall of 98% and a precision of 96% over the test set (composed of 1909 ring boundaries). The code is available, and detections can be exported in the *pos* format for editing with the CooRecorder tool [42]. Wu et a. [60] applied a Feature Pyramid Network (FPN) with the encoder of ResNet-18 for tree ring segmentation in hardwood species, reporting an F-Score of 73%. García-Hidalgo et al. [14] applied a UNETR network to segment tree rings in microscopy images, reporting that their model performed equal or better than manual tree ring boundary delineation on stained wood microsections in 92% of the cases.

Regarding cross-sectional images, Gillert et al. [17] addressed the challenge of detecting tree rings in microscopy images of shrub cross sections by introducing the Iterative Next Boundary Detection (INBD) method, which employs a U-Net-based architecture. The INBD method is assessed in the species *Dryas octopetala L.*, *Empetrum nigrum subsp. hermaphroditum (Hagerup) Böcher*, and *Vaccinium myrtillus L.*. While the source code is publicly available, the method lacks a graphical user interface for editing or correcting ring detections.

In this chapter, we introduce the Tree Ring Analyzer Suite (TRAS), a tool that features a graphical interface and is relatively easy to install. It is designed for the semi-automatic detection of growth rings in cross-sectional images, with an option for manual correction. It incorporates the INBD [17], CS-TRD (Section 4.2), and DeepCS-TRD (Section 4.5) automatic tree ring detection methods, which have been applied to various species, including the trees *P. glauca*, *P. taeda*, and *G. triacanthos*, as well as the shrubs *Salix glauca L.*, Dryas octopetala, Empetrum hermaphroditum, and Vaccinium myrtillus. The tool provides detailed two-dimensional measurements, facilitating an understanding of growth dynamics in tree species over time, and allows manual corrections of the automatically produced results. The semi-automatic character of TRAS significantly accelerates the measurement process of biological growth characteristics on a per-sample basis (see Section 5.5). Moreover, ring measurements made by TRAS are validated with the CooRecorder tool [42].

5.2 Tree Ring Analyzer Suite

The Tree Ring Analyzer Suite (TRAS) is an interactive software for tracing and editing tree rings in cross-sectional disc images. It is fully developed in Python and utilizes

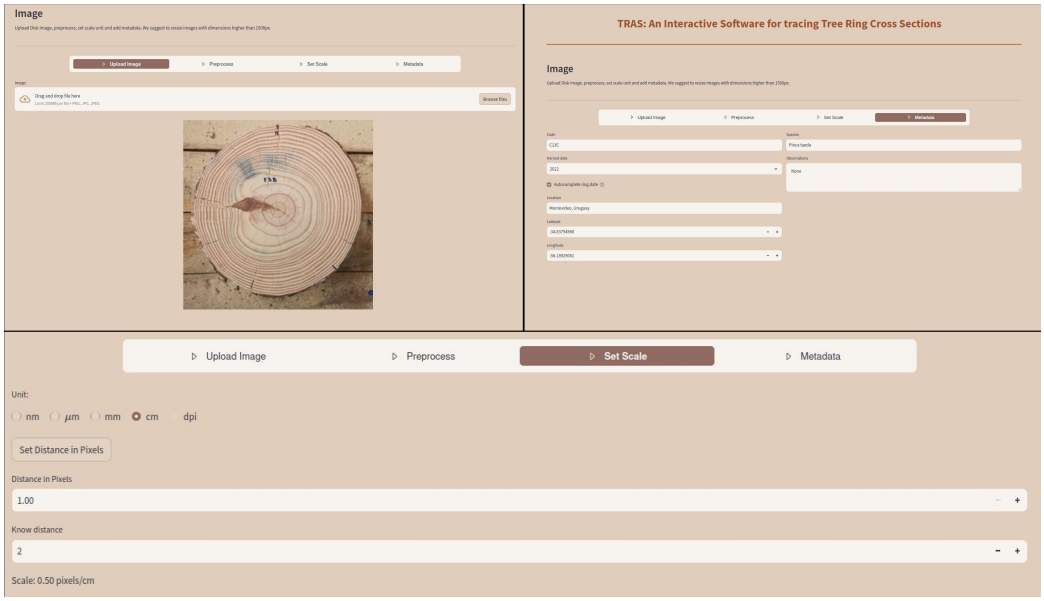


Figure 5.1: TRAS Image menu which allows preprocessing the image, adding relevant metadata, and setting the pixel-to-millimeter ratio

the Streamlit web framework as its frontend. It provides a user-friendly graphical interface that allows general users to automatically obtain area-based measurements or semi-automatically extract them from cross-sectional images. Python was chosen as the base language, allowing developers to easily add improvements to the graphical interface or detection algorithms as needed. TRAS currently incorporates all state-of-the-art automatic algorithms for ring detection in RGB images of tree and shrub cross-sections that we are aware of. The tool is open source under the MIT license, and the code is available at this link.

Image menu TRAS comprises four key steps. The first involves uploading the image, preprocessing it, and adding relevant metadata. Once the acquired image is uploaded to the TRAS system, it undergoes several preprocessing steps to optimize the tree ring detection. Users can remove the background manually or automatically using a U^2Net network [50], which might improve the accuracy of the automated ring detection methods.

Additionally, high-resolution images (with a width greater than 10,000 pixels) can be resized to ensure smooth tool performance. To minimize interpolation artifacts during this process, Lanczos resampling is applied [16]. A scale has to be set to establish the pixel-to-millimeter ratio required for accurate metric computations. Figure 5.1 illustrates the menus for preprocessing the image, adding metadata, and setting the scale.

Ring Detection menu The second step is the automatic detection of the rings (see Figure 5.2). TRAS incorporates three algorithms for automatic ring detection: CS-TRD, DeepCS-TRD, and INBD (Chapter 4). These automatic methods for ring detection need the location of the pith, and the first growth year. TRAS includes three automatic pith detector methods [41]: automatic wood pith detection (APD), PClines-based automatic wood pith detection (APD-PCL), and deep learning based automatic wood pith detection (APD-DL) (see Chapter 3). Pith location can be marked manually if required as well. Ad-

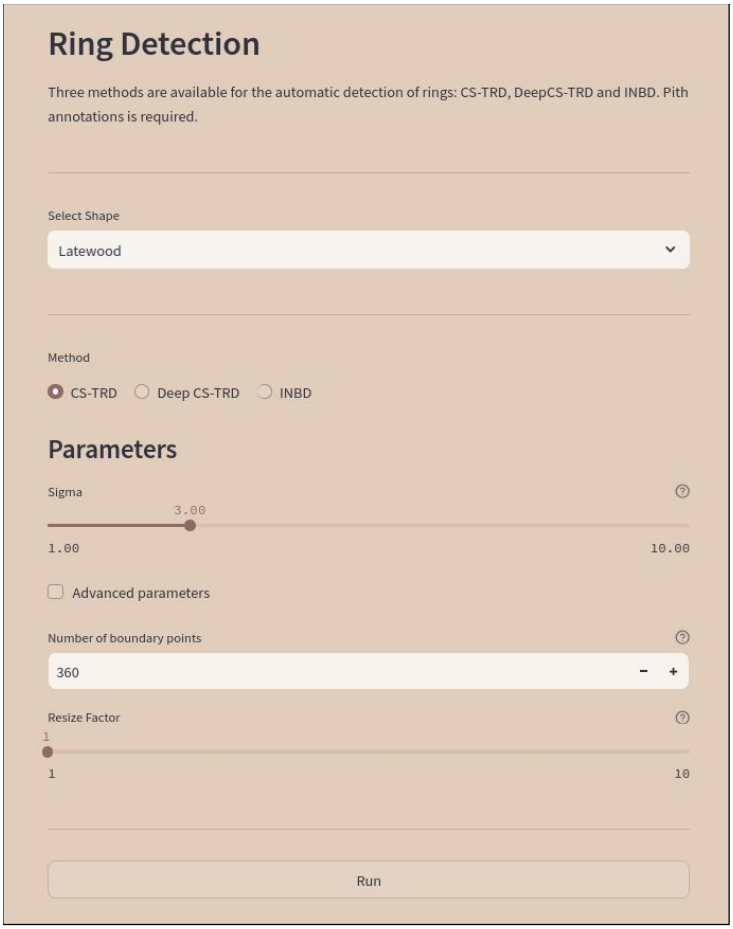


Figure 5.2: Menu for automatic tree ring detection (*Ring Detection*). First, the object to be detected is chosen from the *Select Shape* dropdown menu, with three options available: pith, annual ring boundary, and earlywood-latewood ring boundary. Second, the automatic detection method is selected. For annual ring boundaries, CS-TRD, DeepCS-TRD, and INBD are available. Both deep learning-based methods allow users to upload a custom model or select from predefined models. Finally, two generic options are provided: the number of points, which specifies the number of points used to define each ring boundary, and the resize factor, which determines the resolution of the input image for processing. While the algorithms operate on the resized image, the resulting ring boundaries are mapped back to the original image resolution.

ditionally, the CS-TRD method can automatically detect the earlywood-latewood boundary if required.

Ring Editing menu The third step is the interactive tree ring adjustment. A Python-based graphical user interface, built with the Streamlit web framework, enables user interaction. Predictions generated by the automated method can be refined using the LabelMe tool [59], which is integrated directly within Streamlit. This setup enables users to remove false ring detections, correct existing ones, and add any rings that were not initially detected. The same interface will enable users to manually delineate the tree rings without using automatic tools or delineate the *earlywood-latewood* boundaries. Other shapes,

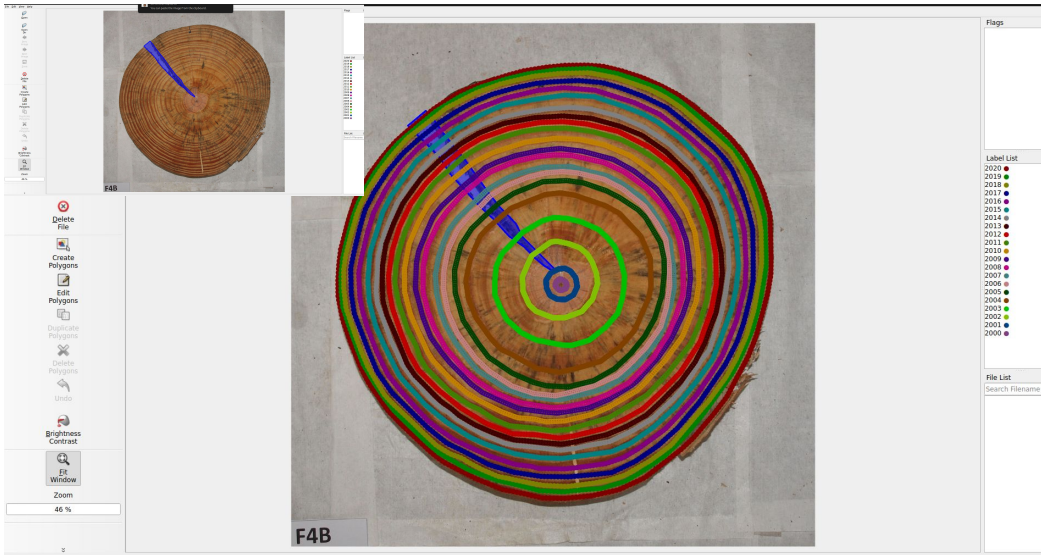


Figure 5.3: Menu for automatic tree ring postprocessing (*Ring Editing*). Rings are represented as polylines with a fixed number of points, which can be moved, deleted, or added as needed. Each polyline is automatically labeled with the corresponding growth year, based on metadata provided (e.g., harvest year). Additionally, other shapes can be annotated; for example, a crack is outlined in blue in the image.

such as cracks or knots, can be manually delineated. Figure 5.3 illustrates the edition panel.

Metrics menu Finally, the user can compute various metrics (such as area and perimeter) related to the annual rings or to the *earlywood–latewood* boundaries. The option to delineate the *earlywood–latewood* boundary is included because several studies have shown that latewood ring width is a better proxy for climate variability than the total ring width [20, 56]. The Shapely library [18] is used to compute the ring properties, such as area or perimeter, from the ring boundary points, Figure 5.4 displays the Metrics menu, where the data can be presented in tabular or graphical format for a more straightforward analysis.

Several indicators can be computed, which include area-based metrics such as *earlywood (EW) area*, *latewood (LW) area*, *cumulative EW area*, and *cumulative ring area*. Figure 5.5 illustrates the concept of cumulative areas. If additional shapes such as cracks, knots, or fungus are delineated, the tool provides an option to compute the ring area while excluding the area occupied by these shapes. For instance, the area of the crack highlighted in Figure 5.3 (blue region) can be subtracted from the ring-growth area as needed.

In addition, the perimeters of EW and LW (annual ring) can be computed. The *equivalent ring width* is defined as

$$\Delta r_i^{eq} = \sqrt{Area_i/\pi} - \sqrt{Area_{i-1}/\pi}$$

where $Area_i$ is the area of *ring i* and Δr_i^{eq} is the *equivalent ring width* of ring i. We use a hypothetical circle with the same area as the ring i.

A *Ring's Circle Similarity Factor* (SF) is also added. This indicator measures how far

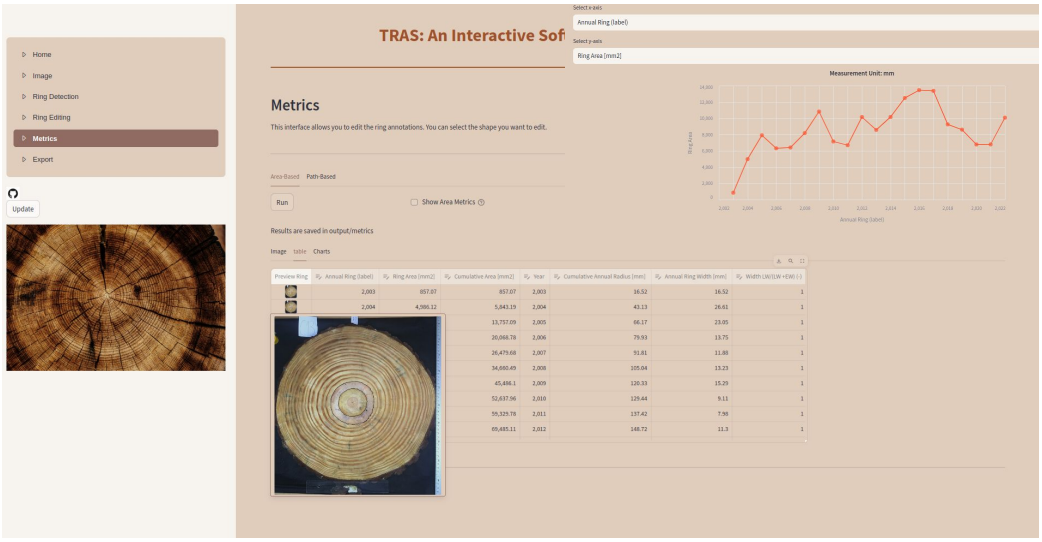


Figure 5.4: Menu for ring metrics visualization. Metrics can be presented in a plot or a table format.

the ring is from a perfect circle (takes values between 0 and 1). It is computed as

$$SF = 1 - \frac{Perimeter_i - 2\sqrt{\pi Area_i}}{Perimeter_i}$$

where $Perimeter_i$ and $Area_i$ are the perimeter and area of *ring i*.

The *Ring Eccentricity* is measured by the difference between the centroid of the cumulative area of the annual ring and the pith. It is a vector; therefore, the module and phase of the eccentricity are reported.

Additionally, a graphical user interface is provided to measure the ring width along a specified path, similar to the one described in [42]. TRAS exports the results in *pos* format, allowing for straightforward comparison with the results of other algorithms or manual measurements.

5.2.1 Installation and usage

To install the tool, Python 3.12 must be available on your system. Execute the following commands, replacing `PYTHON_PATH` with the path to your Python interpreter:

```
$git clone https://github.com/hmarichal93/tras.git
$cd tras
$./install.sh PYTHON_PATH
```

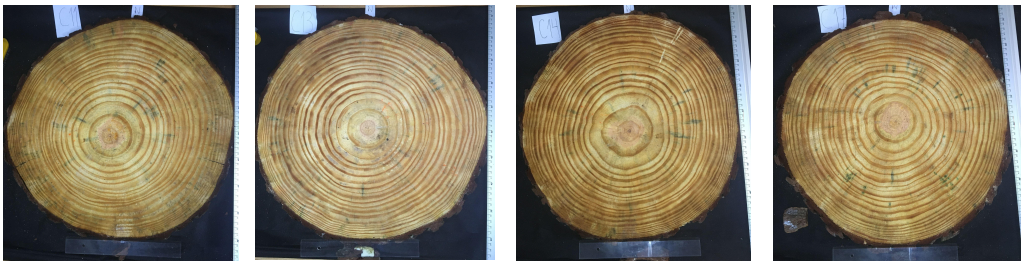
To run the application, execute the command

```
$streamlit run app.py
```

The application was developed and tested on Ubuntu LTS 24.04. Instructions for installing the application on other operating systems, such as Windows (using a hypervisor like VirtualBox), are provided in the `README` file of the repository.

(a) (b) (c)

Figure 5.5: Ring structures. The cumulative EarlyWood (EW) area is equal to Area(Ring) + Area(EW). The cumulative ring area is equal to Area(Ring) + Area(EW) + Area(LW). (a) Cross-section image (b). Cross-section image where Area(EW) is highlighted in green while Area(LW) is highlighted in blue. Ring boundaries are highlighted in black. (c) Zoomed-in image.



(a) (b) (c) (d)

Figure 5.6: *Pinus taeda* discs images from the UruDendro2 dataset. (a) C11C. (b) C13C. (c) C14C. (d) C17C.

5.3 Datasets

A collection of *P. taeda* annotated cross sections

The dataset used to assess the accuracy and measurements of automatic tree-ring detection consists of 18 RGB images of *P. taeda* cross-sections (see Figure 5.6 for examples) which is a subset of the UruDendro2 dataset presented at Chapter 2.

Manual measurements of annual ring widths were taken along the north, south, east, and west directions for the disc samples in Figure 5.6. These measurements serve as a reference for validating our tool's performance, using the outputs from CooRecorder [42] as a benchmark.

Other datasets

Additionally, we applied TRAS to images from other datasets that presented additional challenges, such as cracks, fungal damage, or atypical ring patterns, to further validate the tool. In all cases, ring boundary curves were generated using TRAS:

- UruDendro1: Samples F03d and L02c from the species *P. taeda* (see Figures 5.10 and 5.11).
- UruDendro3a: Sample gt6 from the species *G. triacanthos* (see Figure 5.12).
- DiskoIsland: Sample 1W23S20 from the species *S. glauca* (see Figure 5.13).
- TreeTrace_Douglas [35]: Sample C07c from the species *Douglas fir* (see Figure 5.14).

5.4 Validation and accuracy assessment

Automatic tree ring detection

To quantify the manual effort required to obtain complete and accurate ring delineations (i.e., adding, editing, or deleting rings), we compared the detections produced by the methods integrated in TRAS against expert annotations, as described in Section 5.3. For the neural network–based methods INBD and DeepCS-TRD, we used models trained on the UruDendro1 dataset *P. taeda* image dataset introduced in Chapter 2. The CS-TRD method is used with the default parameters.

We assessed the performance of the automatic ring detection methods by evaluating the number of annual rings correctly identified in the *P. taeda* dataset. All detections were performed through the TRAS web interface. A resize factor of 2 was applied to each image to enhance performance, and the background was automatically removed. The pith location was provided by the APD automatic detection method [41].

As evaluation metrics for the automatic detections, we use *Precision*, *Recall*, and the *F-Score*. A threshold of 90% (denoted as *th_pre*) is used to determine the required overlap between the predicted and ground truth rings for a detection to be considered correct. Further details on the evaluation procedure are provided in Appendix B.1.

Measurements

The main contribution of TRAS is its ability to measure ring areas. However, to our knowledge, no existing tool offers comparable ring area measurements for direct validation against TRAS. To compare, we use CooRecorder [42], a widely recognized tool for generating tree ring width measurements along a segment, a feature also included in TRAS. All measurements were performed on post-processed ring delineations in both cases. The outputs from both tools were compared for cross-validation purposes, and linear correlation was assessed using the Pearson correlation coefficient.

Ring width measurements were taken along four directional paths using TRAS and CooRecorder for four disc samples, C17C, C13C, C11C, and C14C, from UruDendro2. In each path, 21 ring widths were measured, resulting in a dataset of 336 pairs of ring width measurements.

Table 5.1: Automatic tree ring detection in samples of the UruDendro2 dataset with the INBD, CS-TRD, and DeepCS-TRD methods. Each row presents the fully automatic ring prediction for each sample. Precision (**P**), Recall (**R**), and F-Score (**F**) are presented in the columns P, R, and F, respectively. The last row represents the average value for each column.

	INBD			CS-TRD			DeepCS-TRD		
Sample	P	R	F	P	R	F	P	R	F
B3C	76.2	69.6	72.7	76.2	69.6	72.7	90.0	78.3	83.7
B19C	81.8	78.3	80.0	94.7	78.3	85.7	90.0	78.3	83.7
B2C	66.7	60.9	63.6	72.2	56.5	63.4	73.7	60.9	66.7
A29C	13.0	13.0	13.0	90.0	78.3	83.7	90.0	78.3	83.7
C10C	72.7	69.6	71.1	66.7	52.2	58.5	84.2	69.6	76.2
A10C	95.5	91.3	93.3	80.0	69.6	74.4	95.0	82.6	88.4
C11C	85.7	78.3	81.8	84.2	69.6	76.2	70.0	60.9	65.1
C13C	95.2	87.0	90.9	86.4	82.6	84.4	95.0	82.6	88.4
B24C	90.9	87.0	88.9	94.4	73.9	82.9	90.0	78.3	83.7
A4C	77.3	94.4	85.0	81.8	100.0	90.0	81.0	94.4	87.2
A30C	95.5	91.3	93.3	90.9	87.0	88.9	90.5	82.6	86.4
C14C	77.3	73.9	75.6	85.0	73.9	79.1	81.0	73.9	77.3
A7C	90.5	82.6	86.4	94.4	73.9	82.9	75.0	65.2	69.8
B25C	91.3	91.3	91.3	86.4	82.6	84.4	90.5	82.6	86.4
B12C	90.5	82.6	86.4	100.0	78.3	87.8	89.5	73.9	81.0
A9C	76.2	69.6	72.7	75.0	78.3	76.6	85.0	73.9	79.1
C16C	95.5	91.3	93.3	85.0	73.9	79.1	90.5	82.6	86.4
C17C	90.5	82.6	86.4	90.0	78.3	83.7	94.7	78.3	85.7
Average	81.2	77.5	79.2	85.2	75.4	79.7	86.4	76.5	81.0

5.5 Results

5.5.1 Automatic tree ring detection

Table 5.1 presents the average performance of automatic ring detection across 18 *P. taeda* samples (comprising 414 tree rings), using the TRAS web interface. All detections were performed via the *Ring Detection* menu (see Figure 5.2), without any user postprocessing. On average, the three methods showed comparable performance. DeepCS-TRD achieved the highest precision at 86.4%, while INBD recorded the lowest at 81.2%. Regarding recall, INBD outperformed the others with 77.5%, whereas CS-TRD obtained the lowest value at 75.4%. Regarding the F-Score, DeepCS-TRD again achieved the best result with 81.0%, while INBD reached 79.2%.

Execution times for automatic detection varied across methods. CS-TRD completed processing in an average of 21.0 seconds per sample, INBD required 45.7 seconds, and DeepCS-TRD 66.6 seconds. These measurements were obtained on a workstation with an Intel Core i5-10300H processor and 16 GB of RAM (CPU only). When using a GPU (NVIDIA GTX 1650), INBD achieved a significantly faster runtime of 6.7 seconds per

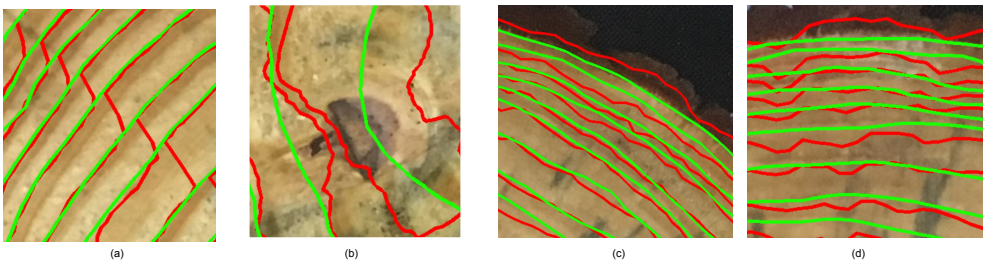


Figure 5.7: Sample B3C from UruDendro2 dataset. The automatic ring detection is depicted in red, and the expert corrected results are shown in green. Some typical errors: a) The wrong ring jump is propagated outwards. b) Knot ring-induced errors. c) and d) lack of precision in the detection.

image, while DeepCS-TRD required 29.5 seconds.

5.5.2 Postprocessing Examples

Figure 5.7 illustrates some cases that required manual postprocessing. The initial automatic ring detection is shown in red, while the corrected detection by an expert is shown in green. A typical error in the INBD method is the propagation of detection errors. Since this method detects rings sequentially from the pith to the bark, a detection error can propagate iteratively to outer rings, as shown in Figure 5.7a. Knots may sometimes interfere with the automatic method, leading to false ring detections (for example, in Figure 5.7b). In some cases, the automatic detections lack precision, as seen in Figure 5.7c and Figure 5.7d.

5.5.3 Measurements

Figure 5.8 illustrates the linear correlation between the measurements made with TRAS and with Coorecorder detailed in Section 5.4 (Pearson's $r > 0.99$, $p < 0.001$, $n = 336$). The linear regression slope is close to 1 (0.9927), and the intercept is nearly 0 (0.0354), indicating strong agreement between the two tools. The root mean square error between both sets of measurements is 0.1478 mm. This experiment indicates that the measurements obtained with TRAS are very similar to those made with Coorecorder, a commonly used tool.

5.6 Discussion

Two-dimensional tree-ring area measurements may exhibit stronger correlations with climatic variables, such as temperature and precipitation, than traditional one-dimensional metrics like ring width [34]. In species such as *Santalum paniculatum*, which exhibit asymmetric growth patterns, two-dimensional ring boundaries have enabled the quantification of growth characteristics that are difficult to capture using conventional ring-width analysis [5]. Computing such two-dimensional metrics from curves that delineate the

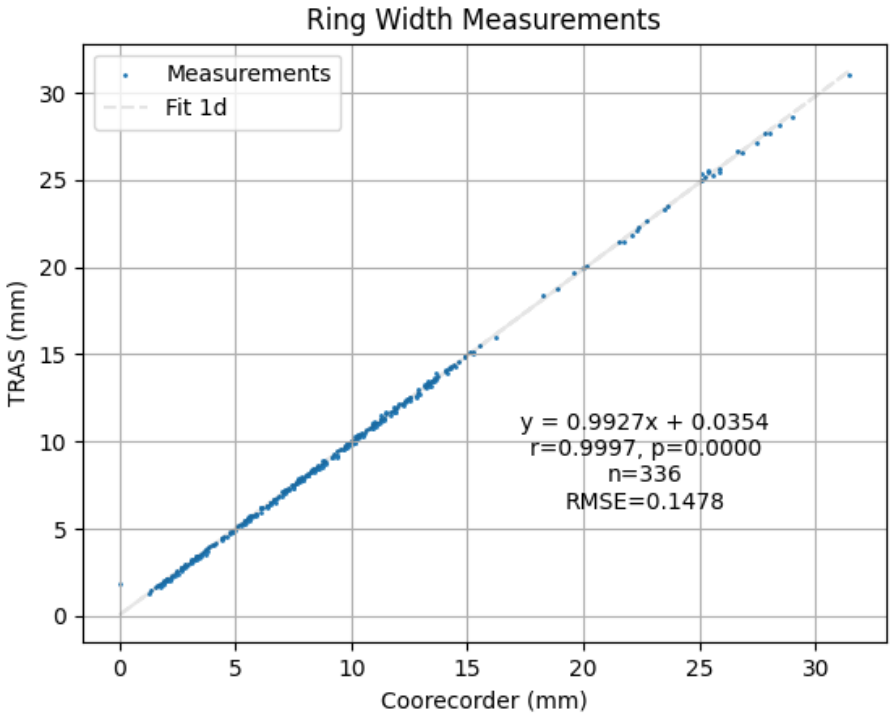


Figure 5.8: Comparison between tree ring width measurements with CooRecorder and TRAS over the test dataset. The linear correlation is apparent between the two measurement methods.

boundaries between growth rings, as proposed by [5], appears to be the most appropriate approach. However, manually generating these curves remains a labor-intensive and time-consuming task.

Automatic ring detection

A graphical interface that integrates automatic ring delineation methods can significantly reduce the manual effort required for ring marking, allowing researchers to focus on analyzing biologically relevant measurements, such as ring area. To our knowledge, there are two open-source approaches for the automatic detection of tree rings. The first is based on a model of the stem, known as the spider-web structure, which is proposed in Section 4.2 and initially applied to conifers. It was later extended in DeepCS-TRD (Section 4.5) to other species, such as *G. triacanthos*. The second is the INBD method introduced by [17], designed for microscopy images of shrubs, which sequentially segments rings from the pith to the bark. Both families of methods rely on post-processing to enforce the concentric structure of growth rings. These approaches have been applied to various species, including the trees *P. glauca*, *P. taeda*, and *G. triacanthos*, as well as the shrubs *S. glauca*, *D. octopetala*, *E. hermaphroditum*, and *V. myrtillus*.

When applied to species with high contrast at ring boundaries, these methods yield

Figure 5.9: *P. taeda* sample C14C from UruDendro2 dataset. (a) Sample after background removal. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

Figure 5.10: *P. taeda* sample F03d from UruDendro1. (a) Sample after background removal. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

excellent results, even in the presence of cracks or fungal infections. Figure 5.9 illustrates the performance of automatic ring detection on a *P. taeda* sample (C14C from the UruDendro2 collection). The red curves represent the automatic detection, while the expert-corrected annotations are shown in green. Most rings are correctly identified, but user interaction is still required to complete the delineation. Specifically, the pith and outermost rings are not detected and must be manually marked. As shown in Figure 5.9d, minor adjustments are necessary for a few rings. Additionally, the automatic method generates four false detections that must be removed.

Automatic ring delineation also performs reliably in the presence of knots, as shown in sample F03d from UruDendro1 dataset (Figure 5.10). The only missing ring corresponds to the pith; all others are correctly detected with minor errors visible in Figure 5.10b and d. Furthermore, most rings are still successfully detected in samples affected by fungal staining, such as L02d in Figure 5.11. Cracks do not significantly impact detection performance either.

Deep learning methods, such as DeepCS-TRD and INBD, are also effective for species with internal color variation, as seen in the angiosperm *G. triacanthos* (see Figure 5.12a), which exhibits darker hardwood regions near the center of the stem. In this case, annual ring boundaries are less distinct than in species like *P. taeda*. Figure 5.12b shows the rings automatically detected by DeepCS-TRD in red, overlapping the ground truth delineations

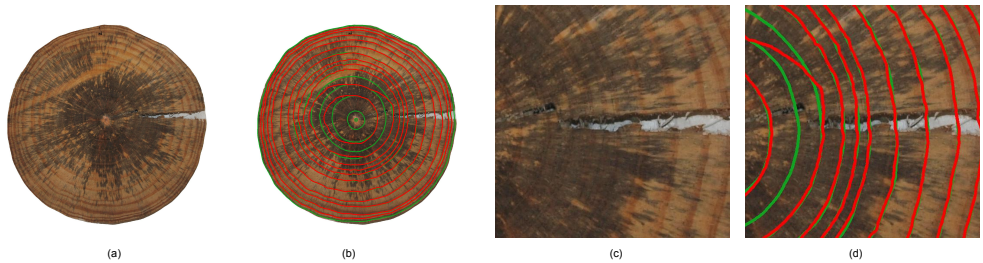


Figure 5.11: *P. taeda* sample L02b from UruDendro1. (a) Sample after background removal. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

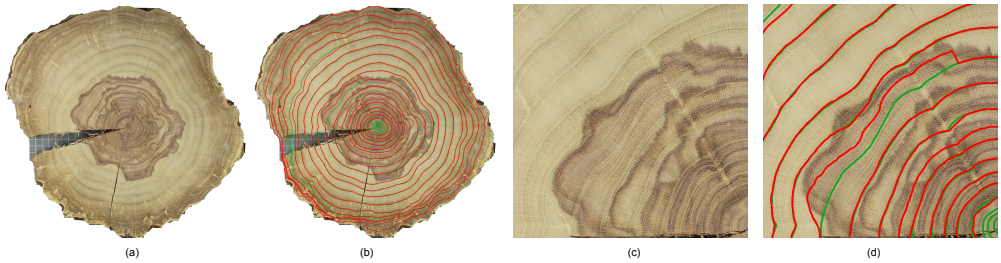


Figure 5.12: *G. triachantos* image sample gt6 from UruDendro3a. (a) Sample after background removal. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

in green. While innermost rings are often missed, as evidenced by a concentration of green curves near the pith. In this case, most rings are correctly detected. With minor corrections, a complete delineation of the disc can be obtained.

U-Net-based approaches have also shown promising results in species where ring boundaries follow distinct anatomical patterns, such as ring-porous structures, [10]. Figure 5.13 presents the results of applying DeepCS-TRD, which internally uses a U-Net model, to microscopy images of *S. glauca*. The method works fine and fails to detect only two rings: the pith and the bark (highlighted in green in Figure 5.13b). According to Chapter 4, INBD yields the best average performance on shrub images among the three evaluated methods (CS-TRD, DeepCS-TRD, and INBD). This is congruent with the fact that this method was specifically developed for that type of image (microscopic images of shrubs).

Although the above examples are based on images of prepared discs, the methods are also effective on unprocessed discs, provided the ring contrast is sufficiently clear. Figure 5.14 shows the application of DeepCS-TRD to a sample where the image domain differs significantly from the training data. Notably, we can see the strong oblique lines in the image, taken in the field, which are the marks of the saw. Despite this, most rings were correctly detected (highlighted in red), demonstrating the model's robustness to other image domains with clear ring patterns.

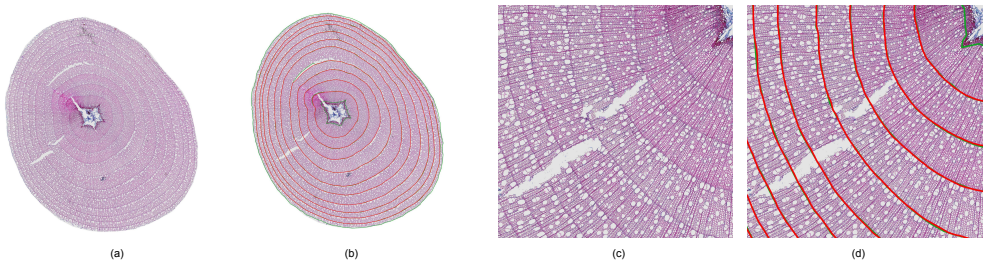


Figure 5.13: *S. glauca* image sample 1W23S20 from DiskoIsland. (a) Sample after background removal. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

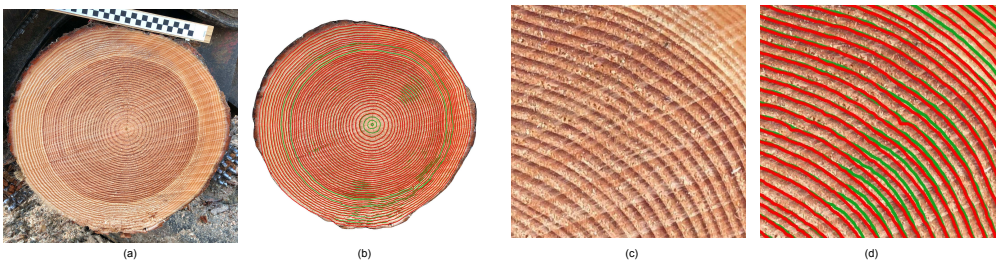


Figure 5.14: *Douglas fir* image sample C07c from TreeTrace_Douglas [35]. (a) Image sample. (b) Expert-delineated rings are shown in green, and automatically detected rings are shown in red. Correct detections overlap with the expert annotations. (c, d) Zoomed-in views highlighting a disc's region.

Finally, TRAS allows users to upload custom models for both DeepCS-TRD and INBD. These models can be trained using ring annotations corrected during post-processing, enabling users to fine-tune detection performance for their specific image datasets and reduce the manual correction required. Details on the training process for adapting the INBD or DeepCS-TRD methods to detect annual rings annotations generated by the TRAS tool can be found in the TRAS GitHub repository.

Postprocessing

Although automatic methods perform well for species with strong ring-boundary contrast (see Figures 5.9 to 5.13), manual postprocessing is often required to obtain accurate measurements. These operations (such as editing, deleting, and adding rings) can be efficiently carried out using the *Ring Editing* panel, as automatically generated rings are represented as editable polylines within TRAS.

Table 5.1 shows that, in 23-year-old samples of the *P. taeda* species, at best 77.5% rings were correctly delineated on average (INBD recall) with a precision of 86.4% at best (DeepCS-TRD), which means there were 22.5% missing rings on average. This highlights the need for tools like TRAS, which integrate both automatic detection methods and postprocessing functionalities to reduce the total time required for ring delineation

while allowing manual correction of misdetections. Moreover, TRAS allows uploading fine-tuned INBD or DeepCS-TRD models to further enhance automatic ring detection. Chapter 4 showed that a training dataset size of only nine samples already gives satisfactory results for the *G. triachantos* species.

Furthermore, TRAS allows for the manual delineation of other structures, such as cracks or knots, to exclude their area from the total ring growth area. An example of this functionality is shown in blue in Figure 5.3.

Limitations

The tool's main limitation is that it is implemented for Linux-based operating systems, whereas most non-developer users typically work in Windows environments. Additionally, the installation process requires the use of the command line and a basic level of technical proficiency, which may pose a barrier for some users.

5.7 Conclusion

We presented TRAS, an open-source graphical user interface tool for automatic tree-ring delineation. It enables users to correct automatic detections and manually compute accurate ring-based measurements. The tool integrates three state-of-the-art automatic detection algorithms (CS-TRD, DeepCS-TRD, and INBD) implemented in Python.

TRAS facilitates the analysis of ring growth in samples where eccentricity limits the applicability of core-based measurements, providing a more comprehensive and efficient alternative. To our knowledge, it is the first graphical tool to combine two-dimensional automatic tree-ring detection with manual correction capabilities on cross-sectional images.

Although TRAS already incorporates all of the functionalities mentioned above, it remains under active development. The long-term goal is to develop a fully cross-platform desktop application that eliminates the need for end-users to interact via the command line.

Esta página ha sido intencionalmente dejada en blanco.

Chapter 6

Conclusions

A new method for automatically delineating tree rings in cross-section images of contrasting tree ring species (*Pinus taeda*) is proposed, referred to as CS-TRD. It is based on a classical Canny-Devarnay edge-detection algorithm to extract ring edge information. After extracting the edges, a postprocessing logic inspired by the biological properties of tree rings is applied to transform them into curves, allowing the computation of ring properties such as area and perimeter. We refer to this postprocessing logic as Spider Web.

This algorithm achieves a performance of 0.787 in mAR and 0.093 in ARAND metrics on the UruDendro1 dataset. On the other hand, the UruDendro2 dataset achieves 0.710 and 0.144 in mAR and ARAND metrics, respectively. These results demonstrate accurate ring delineation, accelerating the precise computation of tree-ring properties.

An adaptation of the CS-TRD method is proposed, where the edge detection step is replaced with a U-Net deep convolutional network. We named this adaptation DeepCS-TRD. This modification allows us to apply the *spider web* model to another species, such as *Gleditsia triachantos* or *Salix glauca*, with different annual ring patterns than for the coniferous species. Additionally, the proposed method increases the performance of CS-TRD also for *Pinus taeda* as can be seen in Table 4.4; in this case, the accuracy (mAR) has improved from 0.787 to 0.906 in UruDendro1 and from 0.710 to 0.832 in UruDendro2. The downside is that this adaptation requires annotated images to train the network.

Additionally, the INBD method for tree-ring delineation in microscopy cross-section images of shrubs was evaluated to identify tree rings in conifer images captured with a smartphone, a distinct application domain. This method is based on the U-Net deep learning architecture and, like DeepCS-TRD, requires annotated data for training. It outperforms the CS-TRD method in all the datasets, particularly in UruDendro1 and UruDendro2, achieving an accuracy (mAR) of 0.846 and 0.742, respectively. In the UruDendro datasets, the INBD pith detector did not achieve satisfactory results.

We proposed two real-time methods for tackling the pith detection problem based on a *spider web* model. They have excellent performance and run in real-time on a CPU-based machine, allowing for a clear comprehension of the approach. The limited number of parameters is understandable and can be fixed once and for all. Additionally, we trained a Yolo V8 architecture for pith detection. It has better (although similar) performance, but has some false negatives and is more opaque concerning the meaning of its millions of parameters.

These automatic methods have been integrated into a graphical user interface, which enables the computation of accurate tree-ring measurements, including area, perimeter, and width. Additionally, it integrates the LabelMe tool to correct the automatic ring detection.

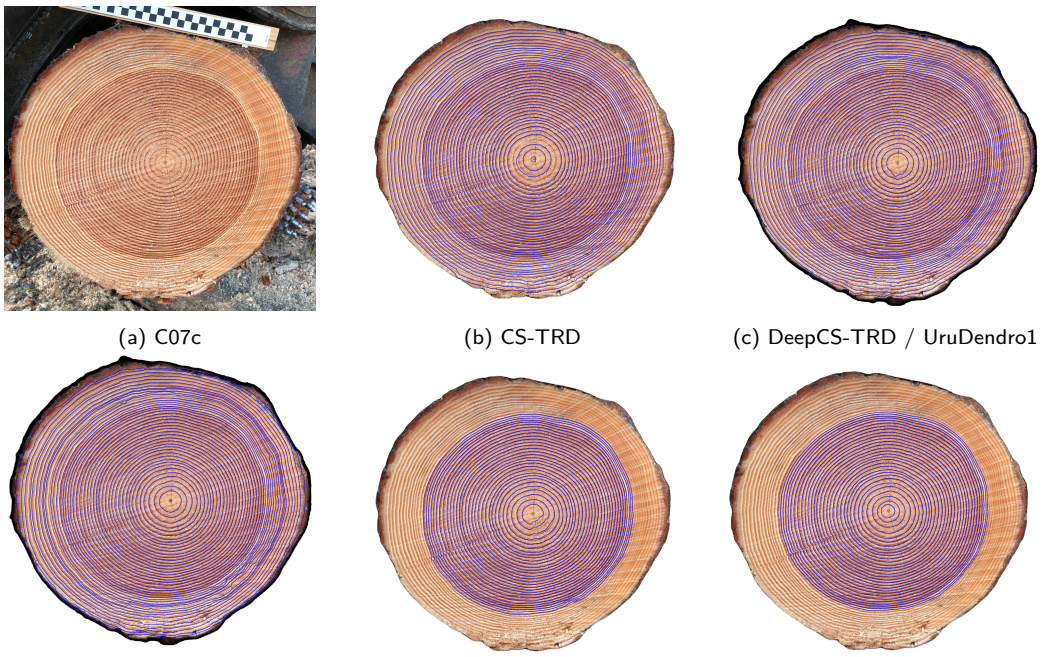
Finally, during this work, we helped to generate and publish tree-ring-annotated datasets for the species *Pinus taeda*, *Gleditsia triacantos*, and *Salix glauca*, which are fundamental to improving automatic tree-ring detection algorithms.

6.1 Future work

During this thesis, a functional prototype of a graphical user interface was implemented, which provides acceptable assistance in computing relevant indicators over tree growth rings. However, it still requires further development to become a fully usable tool in both academic and professional settings. For instance, it cannot yet replace tools such as CooRecorder, which is widely used in dendrochronology and constitutes a highly functional product, despite lacking automated ring detection capabilities.

For commercially important species such as *P. taeda*, one could envision a fully automatic implementation for ring delineation that can be executed in real time on a smartphone in the field. The implementations evaluated in this work have an execution time of approximately 20 seconds on a low-end laptop and already show promising results for automatic ring delineation in cross-sections with clearly visible annual rings. For example, Figure 6.1 and Figure 6.2 illustrate the results obtained with the CS-TRD, deepCS-TRD, and INBD methods on two images of *Douglas fir* acquired in the forest using a smartphone [35]. The deepCS-TRD and INBD models were trained on the UruDendro1 and UruDendro2 datasets. No parameter fine-tuning was performed for the CS-TRD method.

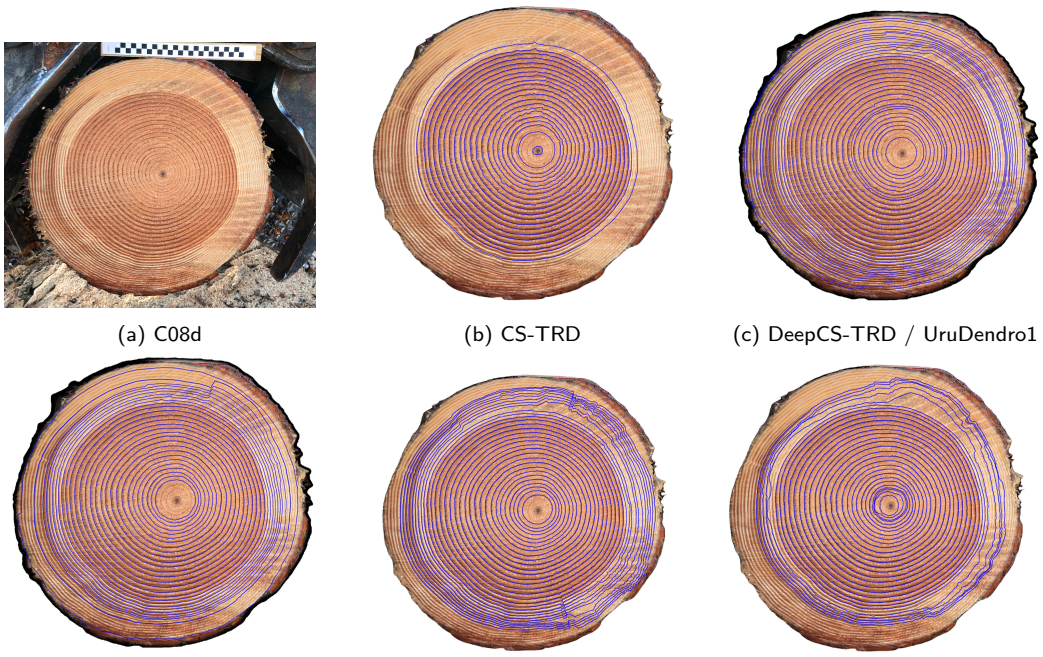
Regarding the species *Gleditsia triacanthos*, further work is needed to expand the dataset to at least 50 images to enable the training of deep learning-based methods.



(a) C07c (b) CS-TRD (c) DeepCS-TRD / UruDendro1

(d) DeepCS-TRD / UruDendro2 (e) INBD / UruDendro2 (f) INBD / UruDendro1

Figure 6.1: C07d sample from *Douglas fir* dataset [35]. Results of different methods, if a dataset appears, it corresponds to the training set.



(a) C08d (b) CS-TRD (c) DeepCS-TRD / UruDendro1

(d) DeepCS-TRD / UruDendro2 (e) INBD / UruDendro2 (f) INBD / UruDendro1

Figure 6.2: C08d sample from *Douglas fir* dataset [35]. Results of different methods, if a dataset appears, it corresponds to the training set.

Esta página ha sido intencionalmente dejada en blanco.

Appendix A

Pith Detection

A.1 Algoritms

lo_sampling Algorithm 13 describe the pseudocode of the local orientation sampling step in the APD Algorithm. The LO estimations are sampled in the following way: 1) ST_C is divided in non-overlapping patches of size $lo_w \times lo_w$ (loops in lines 5 and 6). 2) We find the pixel p^j with the highest coherence (c^j_{high}) within patch $patch_i$ (lines 7, 8 and 9). A minimum patch coherence st_{th} is defined. We assign $ST_O[p^j]$ to $patch_i$ in position p^j, if $c^j_{high} > st_{th}$ (from line 10 to 12) . To fix st_{th}, we calculate the value of ST_C such that a given percentage (parameter $percent_{LO}$) of the LO in the slice has $ST_C > st_{th}$. Each LO is a segment $lo_i = \overline{p^i_1 p^i_2}$, defined by the limits p^i_1 and p^i_2. p^i_{LO} is the middle point between them (p^j). Given the local orientation $\alpha_i = ST_O[p^i_{LO}]$ (line 14), points p^i_1 and p^i_2 are computed as $p^i_{1,2} = p^i_{LO} \pm (\cos(\alpha_i), \sin(\alpha_i))$.

Suppose N patches have coherently enough LO; the output of the step is a matrix, LO_f of size $N \times 4$. In this way, lines are supported by the LO of all meaningful structures in the cross-section, such as the rings.

A.2 APD Parameters

The number of parameters in the machine learning approaches is huge, and they don't have a clear physical meaning. In the classic approaches, such as APD and APD-PCL methods, a limited number of parameters are included, and they have a physical or algorithmic meaning. The APD method has the following parameters, fixed once and for all after a grid search with all the datasets. The default values are in parentheses:

- st_σ: Structure tensor Gaussian σ (1.2).
- st_w: Structure tensor Gaussian kernel size (3 for APD and 7 for APD-PCL).
- $percent_{LO}$: to fix st_{th}, the minimum coherence value to consider valid an LO (0.7).
- lo_w: LO sampling window size (3 for APD and 7 for APD-PCL).

Algorithm 13: lo_sampling

Input: ST_O, // local orientations matrix
ST_C,// coherence matrix
lo_w, //patch size
$percent_{LO}$ // local orientation threshold
Output: A matrix LO_f with the local orientations with higher coherence

$st_{th} \leftarrow$ Calculate the value of ST_C such that a given percentage (parameter $percent_{LO}$) of the LO in the slice has $ST_C > st_{th}$
$i_{max} \leftarrow ST_O$.heigth // lo_w
$j_{max} \leftarrow ST_O$.width // lo_w
$LO_f \leftarrow []$
for $i = 0$ ***to*** i_{max} **do**
  for $j = 0$ ***to*** j_{max} **do**
    $patch_i \leftarrow ST_C$[i.lo_w:(i+1).lo_w,j.lo_w:(j+1).lo_w]
    $p^j \leftarrow$ argmax($patch_i$)
    $c^j_{high} \leftarrow ST_c[p^j]$
    if $c^j_{high} \leq st_{th}$ **then**
      continue
    end
    $p^i_{LO} \leftarrow p^j$
    $\alpha_i \leftarrow ST_O[p^i_{LO}]$
    $p^i_1 \leftarrow p^i_{LO}$ + (cos(α_i), sen(α_i))
    $p^i_2 \leftarrow p^i_{LO}$ - (cos(α_i), sen(α_i))
    $LO_f \leftarrow LO_f$ + (p^i_1, p^i_2)
  end
end
return LO_f

- r_f: a factor used to estimate the side size of the search region for the iteration (7).

The APD-PCL method adds the following:

- *ransac_outlier_th*: RANSAC residual threshold defining the width of the line cluster (0.03).

A.3 Methods comparison: difficult cases

Figure A.1 illustrates how all the presented methods perform over the difficult cases in the UruDendro2 collection. Figure A.1a to A.1c illustrates cases where the ACO method (red marker) performs poorly. In one case, the method even predicts outside the disc region (Figure A.1a). In this cases, the APD, APD-PCL and APD-DL (blue, yellow, and green

(a) B12-2 (b) C17 (c) C10-2

(d) F07e (e) L02a (f) L02b

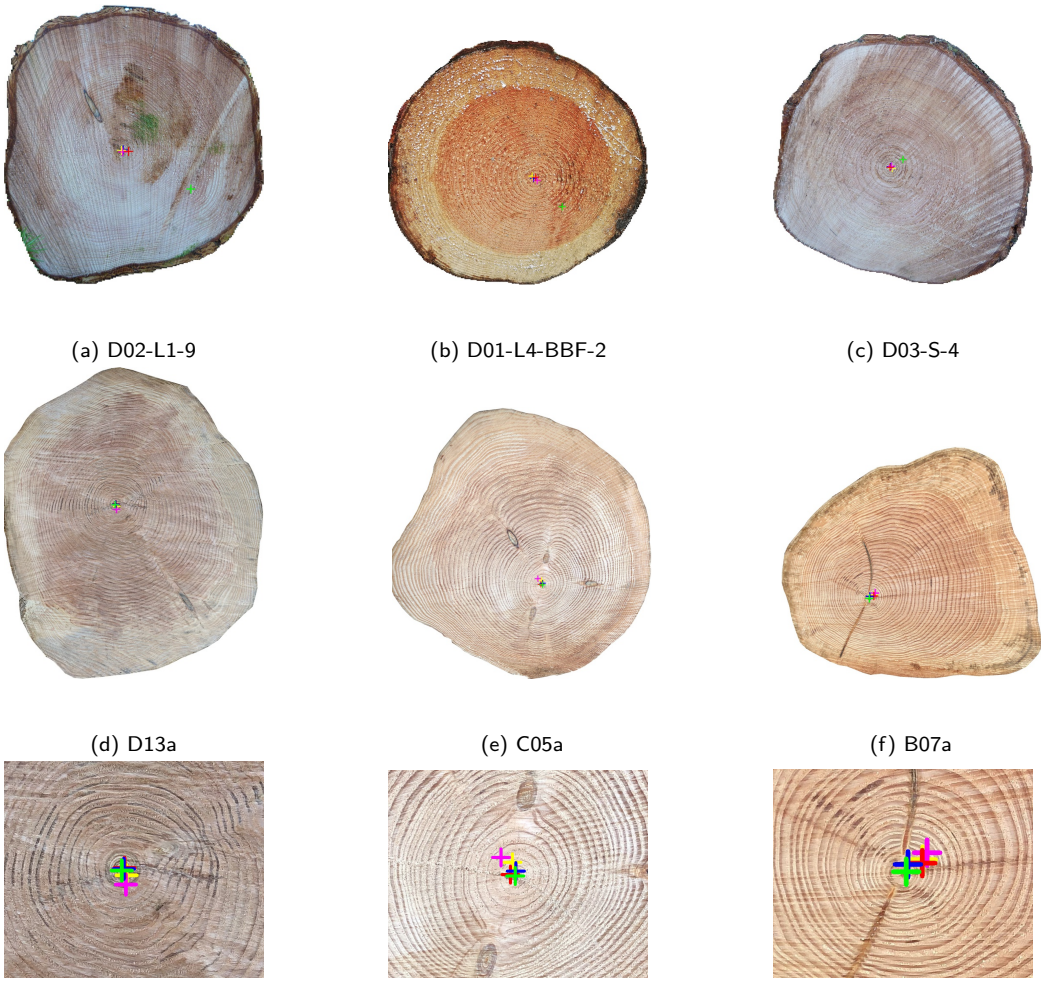
Figure A.1: Results over difficult cases, e.g cracks, fungus presence in Uru2 collection. Purple, LFSA; Red, ACO; Blue, APD; Yellow, APD-PCL and Green, APD-DL

markers, respectively) perform similarly, all of them close enough to the ground truth pith (the prediction is within the first tree ring region). On the other hand, the LFSA method (purple marker) performs slightly worse.

Figure A.1d to A.1f show discs with a strong fungus presence around the pith position. In this case, the APD-PCL (yellow marker) performs considerably better than the other methods (in all cases, the prediction is within the first tree ring region). When there is no tree ring information, the tree ring-based methods (LFSA, ACO and APD) fail and do not converge near the pith position as illustrated in Figure A.1e and A.1f. This is the critical scenario for the ACO and APD methods. In the last stages, they search for the local minimum in a region centered around the previous pith prediction. The ring information will be very limited if the region is too small, which produces a method divergence. In the other hand, the APD-PCL method integrates radial structural insights derived from cracks and fungus in addition to the ring structure. Thereby enhancing its performance significantly in this particular scenario. Finally, the APD-DL method (green marker) also performs poorly in this scenario.

Figure A.2 illustrates cases of off-center pith in Forest, Logyard and Disc collections and the presence of (strong) saw marks. As seen in Figure A.2a to A.2c, the APD-DL method performs considerably worse than the other methods. These disks belong to the Forest and Logyard collections. Both collections have 89 images, a small data size for training a deep learning method. As shown in the *Results* section, both APD and ACO outperform the APD-DL method on this dataset.

Figure A.2d to A.2f, show images from the Discs collection in which the pith is



(a) D02-L1-9 (b) D01-L4-BBF-2 (c) D03-S-4

(d) D13a (e) C05a (f) B07a

(g) Zoom In D13a (h) Zoom In C05a (i) Zoom In B07a

Figure A.2: Results over difficult cases, e.g fungus presence, no pith eccentricity (uncentred pith position) in Forest, Logyard and Disc collections. Purple, LFSA; Red, ACO; Blue, APD; Yellow, APD-PCL and Green, APD-DL

strongly off-centered. A zoom-in is shown for each disc in Figure A.2g to A.2i. On this scenario, the LFSA method (purple marker) is less precise (in all the cases, it is further from the ground truth pith), whereas APD and APD-DL perform similarly.

Appendix B

Tree Ring Detection

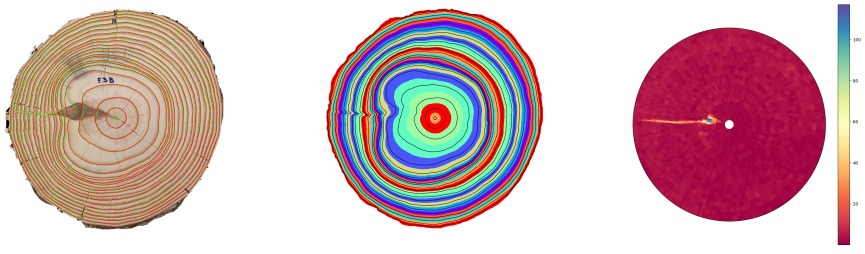
B.1 Metric

To assess the automatic tree-ring delineation methods, we developed a metric based on the one proposed by Kennel et al. [26]. To determine if a ring is detected, we define a ring influence area as the set of pixels closer to that ring. For each ray, the frontier is the midpoint between the nodes of consecutive ground truth (GT) rings. Figure B.1b shows the influence area for rings in disc F03d from Urudendro1 collection. Figure B.1a shows the detections (in red) and GT marks (in green) for the same image.

The influence region associates a detected curve with a GT ring. In both cases, nodes are associated with the N_r rays. Given a GT ring, we assign it to the nearest detection using:

$$Dist = \sqrt{\frac{1}{N_r} \sum_{i=0}^{N_r-1} (dt_i - gt_i)^2} \tag{B.1}$$

Where i represents the ray direction, dt_i is the radial distance of detected node i, and gt_i is the radial distance of the corresponding GT node i. As a reminder, the radial distance



(a) (b) (c)

Figure B.1: Measuring the error of automatic detections for Urudendro1 image F03d: (a) In green, the GT; in red, the detections produced by the method. (b) Areas of influence of the GT rings. (c) Absolute error, in pixels, between the detections and the GT.

of $node_i$ is defined as the Euclidean distance between $node_i$ and the pith (Equation (4.2)).

The closest detection may be far from the corresponding GT ring. To match a detected curve with a GT ring, it is essential to ensure that the identified chain is the closest one to the ring and that it is sufficiently close. We utilize the influence area of each GT ring (Figure B.1b). Upon detecting a curve, if the proportion of nodes within that chain that falls within the influence area of the nearest ring exceeds a specified threshold parameter (th_pre = 60%, see Appendix B.2.2), we assign the detected curve to the corresponding GT ring. If it falls below the threshold, the detection is not associated with any GT ring. In other words, for a detected curve to be assigned to a GT ring and considered a true positive, at least 60% of its nodes must be within the influence area of that GT ring.

We define the absolute error $A\epsilon_i$ (in pixels) for the *node i* as the absolute difference in pixels between the GT ring and the detected ring associated with it:

$$A\epsilon_i = |r_i^d - r_i^{GT}|$$

Where r_i^d is the radial distance from the center to node i of the detected ring, and r_i^{GT} is the same for the GT ring. Figure B.1c shows the absolute error between the GT and the detected rings assigned to them. Red represents a low error, while yellow, green, and blue represent a higher error. As can be seen, the error is concentrated around the knot, affecting the precise detection of some rings.

Once all the detected chains are matched with the GT rings, we calculate the following values:

1. True Positive (TP): when the identified closed chain and the GT ring match.
2. False Positive (FP): when the identified closed chain doesn't match any GT ring.
3. False Negative (FN): when a GT ring doesn't match any detected closed chain.

Precision is given by $P = \frac{TP}{TP+FP}$, Recall by $R = \frac{TP}{TP+FN}$ and the F-Score by $F = \frac{2PR}{P+R}$. Table B.1 shows the assessment of each tree ring detection method using the metric presented in this section.

B.2 Experiments

This section presents some experiments to help us better understand the methods and their limitations.

B.2.1 Pith position sensibility

In this section, we assess the CS-TRD's sensitivity to errors in the pith estimation. Figure B.2a shows 48 different pith positions used in this experiment (eight different pith positions across six rays). These radially displaced pith positions are selected as follows:

- We define an error step along a ray as 25% of the first ring equivalent diameter.
- Three positions are marked inside ring 1, with errors 25%, 50%, and 75% off the GT center in the ray direction.

Table B.1: Tree ring methods assessment with the UruDendro metric. All dataset images were resized to 1504x1504 pixels. The highest result per dataset is in bold.

Dataset	**P** (↑)	**R** (↑)	**F1** (↑)	**RMSE** (↓)	**Method**
Uru1	0.963	0.910	0.931	2.717	CS-TRD
Uru2	0.908	0.880	0.892	3.005	
Uru3a	0.714	0.500	0.588	10.694	
Disko	0.384	0.180	0.227	27.934	
Uru1	**0.986**	**0.938**	**0.961**	**1.209**	DeepCS-TRD
Uru2	**0.939**	**0.895**	**0.916**	**1.734**	
Uru3a	**0.652**	**0.750**	**0.698**	**3.006**	
Disko	0.854	0.815	0.830	7.251	
Uru1	0.964	0.929	0.945	2.353	INBD
Uru2	0.866	0.893	0.879	3.300	
Uru3a	0.609	0.700	0.651	14.925	
Disko	**0.904**	**0.906**	**0.905**	**6.727**	

- One position is marked on ring 1.
- Three positions are marked between the first and second rings, with the same increasing errors in the ray direction.
- Finally, another position is marked on ring 2.

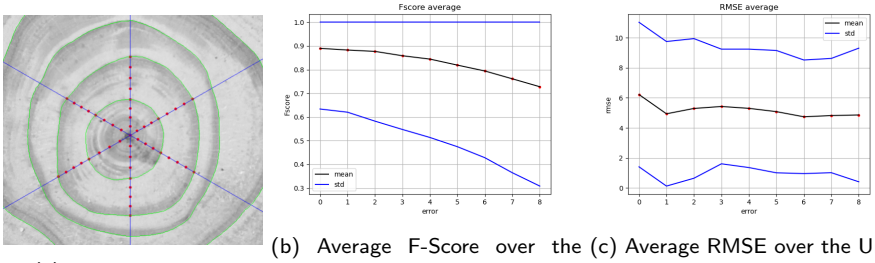
We executed the algorithm for each disc and pith position of the UruDendro1 dataset ($\sigma = 3.0$), resulting in 48 outcomes. We calculated the average RMSE and F-Score (see Appendix B.1) measurements for the six-ray directions for each radially displaced pith position. This produced two eight-coordinate vectors, one for RMSE and one for the F-Score. Figures B.2b and B.2c illustrate the average F-Score and RMSE for each error position across the dataset, respectively. The F-Score decreases as the error in the pith estimation increases, while the RMSE is less sensitive to pith errors.

B.2.2 Detection-to-ground-truth assignation threshold

In this experiment, we examine how performance changes with different values of the *th_pre* parameter, which determines the number of ring nodes within the influence area to be considered in the detection-to-ground-truth assignation step. Figure B.3 displays the results for the UruDendro1 and Kennel datasets. As anticipated, higher precision leads to lower RMSE but lower F-Score. Based on these findings, we fixed *th_pre* = 60% as the default value, which appears to be a good compromise.

B.2.3 CS-TRD, edge detector optimization stage

The algorithm heavily relies on the edge detector stage. In this experiment, we test different σ values for the Canny Devernay edge detector to get the one that maximizes the F-Score for the UruDendro1 dataset. This dataset shows significant variations in image



(a) Pith position grid (b) Average F-Score over the UruDendro1 dataset. (c) Average RMSE over the UruDendro1 dataset.

Figure B.2: Pith position experiment. (a) Eight different pith positions are marked on six ray directions. We executed CS-TRD for each pith position. GT rings are in green. (b and c) For each disc of the UruDendro1 dataset, we run the method using the 48 different pith positions. Results are averaged over the six rays' directions per error position. The black curve represents the mean, while the blue curve represents the standard deviation.

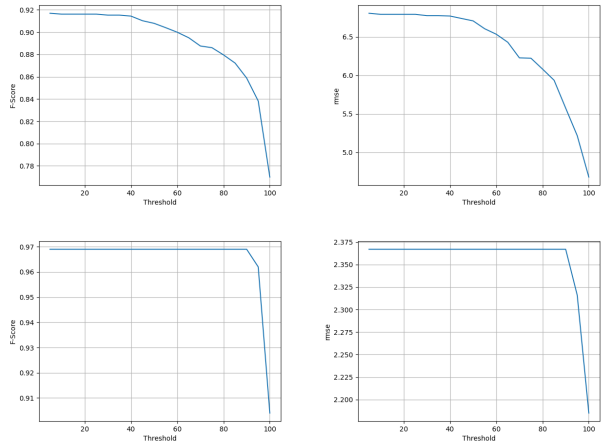


Figure B.3: Performance metrics were computed for different values of the *th_pre* parameter. The first row shows the results for the Urudendro1 dataset, and the second row for the Kennel dataset. The left column shows the average F-Score, and the right column shows the average RMSE.

resolution, allowing us to study the overall performance with different input image dimensions. Results are presented in Figures B.4a and B.4b. We compute the average F-Score for the original image sizes and then scale all images in the dataset to 640×640, 1000×1000, and 1500×1500 pixels. The best result was achieved for size 1500×1500 with $\sigma = 3.0$. Execution time varies with image size. The average execution time for this size is 17 seconds. The execution time decreases as σ increases because fewer edge chains are detected. Results for the same experiment on the Kennel et al. [26] dataset are shown in Figures B.4c and B.4d. The best F-Score is achieved for 1500×1500 size with $\sigma = 2.5$. The lower optimal σ value can be attributed to the Kennel dataset having images with more rings, averaging 30 rings per disc, while the UruDendro 1 dataset has 19 rings per disc on average.

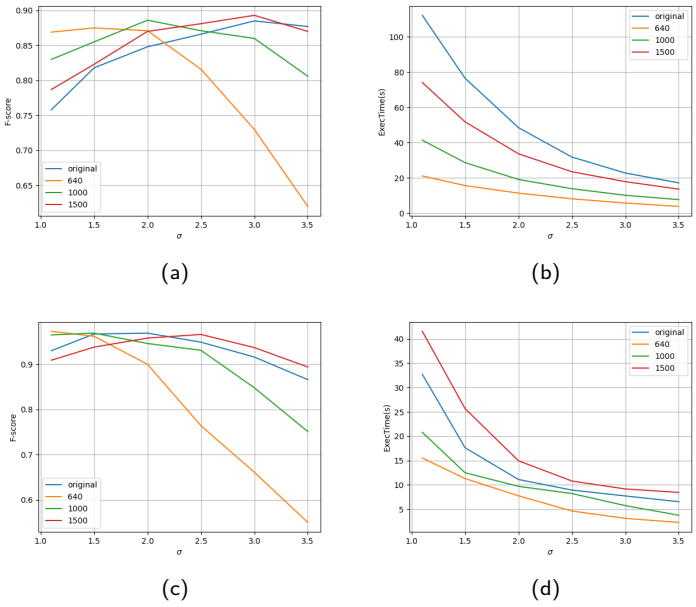


Figure B.4: Influence of the image size and edge detector σ parameter experiment. Each curve represents a different image resolution: 640×640, 1000×1000, 1500×1500, and the original resolution (in blue). (a) Average F1 vs. σ for different image sizes of the UruDendro dataset. (b) Average execution time (in seconds) vs σ for different image sizes over the UruDendro1 dataset. (c) Average F1 vs. σ for different image sizes of the Kennel dataset. (d) Average execution time (in seconds) vs σ for different image sizes over the Kennel dataset.

B.2.4 DeepCS-TRD, additional experiments

Section 4.8 presents the automatic tree detection results on each dataset for the assessed method using a fixed set of parameters. Not using tile (tile_size = 0) improves the performance in the UruDendro1 dataset from 0.906 to 0.923 in mAR and from 0.051 to 0.045 in ARAND as it is illustrated in Table B.2. Moreover, execution time improved as well from 22.5 to 12.4 seconds. A similar improvement is obtained in the UruDendro2 dataset from 0.832 to 0.846 in mAR and from 0.89 to 0.080 in ARAND. Execution time decreased as well from 29.5 to 16.7 seconds.

On the other hand, the performance of DeepCS-TRD on the UruDendro3a and DiskoIslanda datasets decreased when tiles were not used, as shown in Table B.2. Using an overlapping-tile strategy allows us to increase the training dataset size, which could explain the higher performance when it is used.

B.2.5 Spider Web additional experiments

In this section, additional experiments are conducted to justify the decisions made during the *spider web* design. CS-TRD method is used to compute the automatic tree ring detection, and the metric presented at Appendix B.1 is used to compare the different methods.

Table B.2: DeepCS-TRD results with tile_size 0 and 256. The value 0 means that the U-Net network was trained with the whole sample image. Additionally, a 0 value means that during inference, the image sample is not divided into patches. Execution time is presented in seconds. ↑(↓) means higher (lower) is better.

Dataset	**mAR** (↑)	**ARAND** (↓)	**exec time** (↓)
Uru1 (0)	**.923**	**.045**	**12.4**
Uru1 (256)	.906	.051	22.5
Uru2 (0)	**.846**	**.080**	**16.7**
Uru2 (256)	.832	.089	29.5
Uru3a (0)	.467	.250	62.8
Uru3a (256)	**.640**	**.116**	**62.6**
Disko (0)	.498	.202	**16.8**
Disko (256)	**.616**	**.116**	21.2

Table B.3: Method performance (average) including the artificial chains with the optimal σ and image resolution over the full UruDendro1 dataset (64 images).

Dataset	**Image Size (pixels)**	σ	**P**	**R**	**F**	**RMSE**
No Artificial Chains	1500x1500	3.0	82.7	66.7	72.1	2.85
Artificial Chains	1500x1500	3.0	92.6	86.3	**89.1**	3.81

Artificial chains

One experiment concerns artificial chains (disc border and pith chain). Table B.3 shows the method performance without including the Artificial Chains and using them. The average metric for the 64 images is presented for both experiments. Adding them improves the results: the F-Score increases from 72.1% to 89.1%.

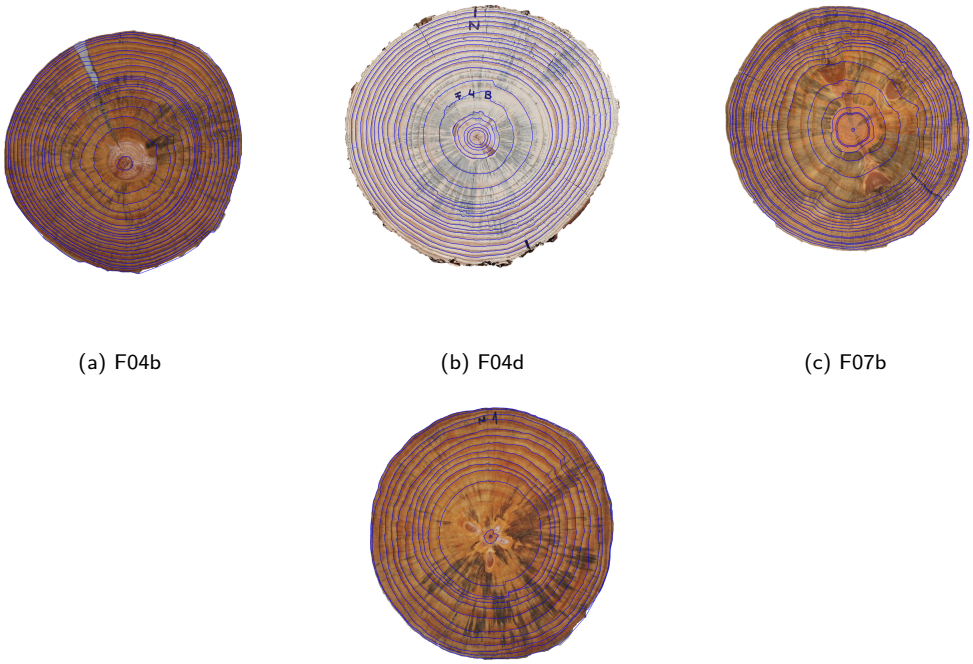
This difference is because chain connections depend on having support in the vicinity. This condition is not met in discs with a strong presence of cracks and/or fungi. In Figure B.5, cases are shown where chains are only completed after adding the artificial chains. Without them, no rings are completed.

Connectivity Goodness Condition

The Connectivity Goodness condition was defined as

$$\textbf{not}\mathit{ExistChainOverlapping} \wedge \mathit{RegularDeriv} \wedge (\mathit{SimilarRadialDist} \vee \mathit{RadialTol})$$

In this section, we explain why the condition $\mathit{SimilarRadialDist} \vee \mathit{RadialTol}$ was chosen over $\mathit{SimilarRadialDist} \wedge \mathit{RadialTol}$. Table B.4 compares the method's performance on the full dataset under both conditions. The equation presented in the main article achieves the best performance, as measured by the F-Score metric.



(a) F04b (b) F04d (c) F07b

(d) L08b

Figure B.5: Samples where closed ring chains are generated after adding the artificial chains.

Table B.4: Method performance (average) comparing the $\wedge$ and $\vee$ conditions in Appendix B.2.5 at the optimal σ and image resolution over the full UruDendro1 dataset (64 images).

Dataset	**Image Size (pixels)**	σ	**P**	**R**	**F**	**RMSE**
SimilarRadialDist $\vee$ *RadialTol*	1500x1500	3.0	94.4	84.0	88.2	3.40
SimilarRadialDist $\wedge$ *RadialTol*	1500x1500	3.0	92.6	86.3	**89.1**	3.81

Connectivity Parameters

As the main text explains, the connect stage iterates its parameters over nine configurations (Table 1 in the main text). Table B.5 shows the results if the Algorithm 7 exits at iterations 1, 2, 7, or 9. In all the cases, artificial chains are included in the last iteration. With nine iterations, we achieve the best performance, characterized by the highest Precision, Recall, and F-Score.

Table B.5: Method overall performance over the full UruDendro1 dataset

iteration (i)	**P**	**R**	**F1**	**RMSE**	**TP**	**FP**	**FN**
1	0.92	0.86	0.88	4.22	16.42	1.41	2.66
3	0.93	0.86	0.89	3.89	16.56	1.16	2.52
7	0.92	0.86	0.89	3.45	16.39	1.19	2.69
9	0.93	0.86	0.89	3.81	16.68	0.84	2.39

Esta página ha sido intencionalmente dejada en blanco.

Bibliography

[1] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cireşan, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9, 2015.

[2] J. Bigun, G.H. Granlund, and J. Wiklund. Multidimensional orientation estimation with applications to texture analysis and optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):775–790, 1991.

[3] G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000).

[4] Mauricio Cerda, Nancy Hitschfeld-Kahler, and Domingo Mery. Robust tree-ring detection. In Domingo Mery and Luis Rueda, editors, *Advances in Image and Video Technology, Second Pacific Rim Symposium, PSIVT 2007, Santiago, Chile, December 17-19, 2007, Proceedings*, volume 4872 of *Lecture Notes in Computer Science*, pages 575–585. Springer, 2007.

[5] Brook M. Constantz, Andrew A. Port, and Randall S. Senock. Comparing automatically generated and manually measured tree-ring transects of growth trends with hawaiian sandalwood as an example species. *Dendrochronologia*, 68:125831, aug 2021.

[6] Rémi Decelle, Phuc Ngo, Isabelle Debled-Rennesson, Frédéric Mothe, and Fleur Longuetaud. Ant Colony Optimization for Estimating Pith Position on Images of Tree Log Ends. *Image Processing On Line*, 12:558–581, 2022. `https://doi.org/10.5201/ipol.2022.338`.

[7] Frédéric Devernay. A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical report, INRIA RESEARCH REP. 2724, SOPHIAANTIPOLIS, 1995.

[8] Marketa Dubska, Adam Herout, and Jiri Havel. Pclines — line detection using parallel coordinates. In *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1489 – 1494, 07 2011.

[9] Philipp Duncker. *Detection and Grading of Compression Wood*, pages 201–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[10] Anna Fabijańska and Małgorzata Danek. Deepdendro – a tree rings detector based on a deep convolutional neural network. *Computers and Electronics in Agriculture*, 150:353–363, 2018.

[11] Anna Fabijańska, Małgorzata Danek, Joanna Barniak, and Adam Piórkowski. Towards automatic tree rings detection in images of scanned wood samples. *Computers and Electronics in Agriculture*, 140:279–289, 2017.

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[13] Miguel García-Hidalgo, Ángel García-Pedrero, Daniel Colón, Gabriel Sangüesa-Barreda, Ana I. García-Cervigón, Juan López-Molina, Héctor Hernández-Alonso, Vicente Rozas, José Miguel Olano, and Víctor Alonso-Gómez. Capturing: A do-it-yourself tool for wood sample digitization. *Methods in Ecology and Evolution*, 13(6):1185–1191, 2022.

[14] Miguel García-Hidalgo, Ángel García-Pedrero, Vicente Rozas, Gabriel Sanguesa-Barreda, Ana I. García-Cervigón, Giulia Resente, Martin Wilmking, and José Miguel Olano. Tree ring segmentation using unet transformer neural network on stained microsections for quantitative wood anatomy. *Frontiers in Plant Science*, 14, 2024.

[15] Rado Gazo, Juraj Vanek, Michel Abdul_Massih, and Bedrich Benes. A fast pith detection for computed tomography scanned hardwood logs. *Computers and Electronics in Agriculture*, 170:105107, 2020.

[16] Pascal Getreuer. Linear methods for image interpolation. *Image Processing On Line*, 1:238–259, 2011. `https://doi.org/10.5201/ipol.2011.g_lmii`.

[17] Alexander Gillert, Giulia Resente, Alba Anadon-Rosell, Martin Wilmking, and Uwe Freiherr Von Lukas. Iterative next boundary detection for instance segmentation of tree rings in microscopy images of shrub cross sections. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14540–14548, 2023.

[18] Sean Gillies, Casper van der Wel, Joris Van den Bossche, Mike W. Taves, Joshua Arnott, Brendan C. Ward, and others. Shapely, oct 2023.

[19] G. H. Granlund and H. Knutsson. *Signal processing for computer vision*. Kluwer Academic Publishers, 1995. ISBN 978-0-7923-9530-0.

[20] Daniel Griffin, Connie A. Woodhouse, David M. Meko, David W. Stahle, Holly L. Faulstich, Carlos Carrillo, Ramzi Touchan, Christopher L. Castro, and Steven W.

Leavitt. North american monsoon precipitation reconstructed from tree-ring latewood. *Geophysical Research Letters*, 40(5):954–958, 2013.

[21] Rafael Grompone von Gioi and Gregory Randall. A sub-pixel edge detector: an implementation of the canny/devernay algorithm. *Image Processing On Line*, 7:347–372, 2017. https://doi.org/10.5201/ipol.2017.216.

[22] Michael Henke and Branislav Sloboda. Semiautomatic tree ring segmentation using active contours and an optimised gradient operator. *Forestry Journal*, 60:185 – 190, 2014.

[23] Jan Hosang, Rodrigo Benenson, Piotr Dollar, and Bernt Schiele. What makes for effective detection proposals? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(4):814–830, apr 2016.

[24] INIA. Open database: Meteorological data. tacuarembo station, 2023. http://www.inia.uy/gras.

[25] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023.

[26] Pol Kennel, Philippe Borianne, and Gérard Subsol. An automated method for tree-ring delineation based on active contours guided by DT-CWT complex coefficients in photographic images: Application to abies alba wood slice images. *Comput. Electron. Agric.*, 118:204–214, 2015.

[27] Donghyeon Kim, ChiUng Ko, and Donggeun Kim. Method for detecting tree ring boundary in conifers and broadleaf trees using mask r-cnn and linear interpolation. *Dendrochronologia*, 79:126088, jun 2023.

[28] Masanari Kimura. *Understanding Test-Time Augmentation*, pages 558–569. Springer International Publishing, 2021.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations, (ICLR), 2015.

[30] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.

[31] Wattanapong Kurdthongmee. A comparative study of the effectiveness of using popular dnn object detection algorithms for pith detection in cross-sectional images of parawood. *Heliyon*, 6(2):e03480, 2020.

[32] Wattanapong Kurdthongmee and Korrakot Suwannarat. Locating wood pith in a wood stem cross sectional image using yolo object detection. In *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 1–6, 2019.

[33] Wattanapong Kurdthongmee, Korrakot Suwannarat, Praepaka Panyuen, and Naruedom Sae-Ma. A fast algorithm to approximate the pith location of rubberwood timber from a normal camera image. In *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–6, 2018.

[34] Julie-Pascale Labrecque-Foy, Sandra Angers-Blondin, Pascale Ropars, Martin Simard, and Stéphane Boudreau. The Use of Basal Area Increment to Preserve the Multi-Decadal Climatic Signal in Shrub Growth Ring Chronologies: A Case Study of *Betula glandulosa* in a Rapidly Warming Environment. *Atmosphere*, 14 (2013).

[35] Fleur Longuetaud, Guillaume Pot, Frédéric Mothe, Alexis Barthelemy, Rémi Decelle, Florian Delconte, Xihe Ge, Grégoire Guillaume, Théo Mancini, Tojo Ravoajanahary, Jean-Claude Butaud, Robert Collet, Isabelle Debled-Rennesson, Bertrand Marcon, Phuc Ngo, Benjamin Roux, and Joffrey Viguier. Traceability and quality assessment of Douglas fir (Pseudotsuga menziesii (Mirb.) Franco) logs: the TreeTrace_Douglas database. *Annals of Forest Science*, 79(46), 2022.

[36] Fleur Longuetaud, Rudolf Schraml, Frédéric Mothe, Tojo Ravoajanahary, Thiéry Constant Rémi Decelle, Phuc Ngo, Isabelle Debled-Rennesson, Karl Entacher, Alexander Petutschnigg, Franka Brüchert, and Andreas Uhl. Traceability and quality assessment of norway spruce (picea abies (l.) h.karst.) logs: the treetrace_spruce database. *Annals of Forest Science*, 80(1), feb 2023.

[37] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. International Conference on Learning Representations (ICLR), 2017.

[38] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[39] Kayla Makela, Tim Ophelders, Michelle Quigley, Elizabeth Munch, Daniel Chitwood, and Asia Dowtin. Automatic tree ring detection using jacobi sets, 2020.

[40] Henry Marichal, Diego Passarella, Christine Lucas, Ludmila Profumo, Veronica Casaravilla, Maria Noel Rocha Galli, Serrana Ambite, and Gregory Randall. UruDendro: An Uruguayan Disk Wood Database For Image Processing. `https://iie.fing.edu.uy/proyectos/madera`, 2023.

[41] Henry Marichal, Diego Passarella, and Gregory Randall. Automatic wood pith detector: Local orientation estimation and robust accumulation. In Apostolos Antonacopoulos, Subhasis Chaudhuri, Rama Chellappa, Cheng-Lin Liu, Saumik Bhattacharya, and Umapada Pal, editors, *Pattern Recognition*, pages 1–15, Cham, 2025. Springer Nature Switzerland.

[42] R. Stockton Maxwell and Lars-Ake Larsson. Measuring tree-ring widths using the coorecorder software application. *Dendrochronologia*, 67:125841, jun 2021.

[43] Sergio Nesmachnow and Santiago Iturriaga. Cluster-uy: Collaborative scientific high performance computing in uruguay. In Moisés Torres and Jaime Klapp, editors, *Supercomputing*, pages 188–202, Cham, 2019. Springer International Publishing.

[44] Kristin Norell. An automatic method for counting annual rings in noisy sawmill images. In Pasquale Foggia, Carlo Sansone, and Mario Vento, editors, *Image Analysis and Processing – ICIAP 2009*, pages 307–316, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[45] Kristin Norell and Gunilla Borgefors. Estimation of pith position in untreated log ends in sawmill environments. *Computers and Electronics in Agriculture*, 63(2):155–167, 2008.

[46] Kristin Norell, Joakim Lindblad, and Stina Svensson. Grey weighted polar distance transform for outlining circular and approximately circular objects. *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pages 647–652, 2007.

[47] Miroslav Poláček, Alexis Arizpe, Patrick Hüther, Lisa Weidlich, Sonja Steindl, and Kelly Swarts. Automation of tree-ring detection and measurements using deep learning. *Methods in Ecology and Evolution*, 14(9):2233–2242, jul 2023.

[48] Candice C. Power, Jakob J. Assmann, Angela L. Prendin, Urs A. Treier, Jeffrey T. Kerby, and Signe Normand. Improving ecological insights from dendroecological studies of arctic shrub dynamics: Research gaps and potential solutions. *Science of The Total Environment*, 849:158508, 2022.

[49] Candice C. Power, Signe Normand, Georg von Arx, Bo Elberling, Derek Corcoran, Amanda B. Krog, Nana Knakkergaard Bouvin, Urs Albert Treier, Andreas Westergaard-Nielsen, Yijing Liu, and Angela L. Prendin. No effect of snow on shrub xylem traits: Insights from a snow-manipulation experiment on disko island, greenland. *Science of The Total Environment*, 916:169896, 2024.

[50] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, and Martin Jagersand. U2-Net: Going deeper with nested U-structure for salient object detection. *Pattern Recognition*, 106:107404, 2020.

[51] Tim Rademacher, Bijan Seyednasrollah, David Basler, Jian Cheng, Tessa Mandra, Elise Miller, Zuid Lin, David A. Orwig, Neil Pederson, Hanspeter Pfister, Donglai Wei, Li Yao, and Andrew D. Richardson. The wood image analysis and dataset (wiad): Open-access visual analysis tools to advance the ecological data revolution. *Methods in Ecology and Evolution*, 12(12):2379–2387, 2021.

[52] Luca Rettenberger, Friedrich Rieken Münke, Roman Bruch, and Markus Reischl. Mask R-CNN Outperforms U-Net in Instance Segmentation for Overlapping Cells. *Current Directions in Biomedical Engineering*, 9(1):335–338, 2023.

[53] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[54] Rudolf Schraml and Andreas Uhl. Pith estimation on rough log end images using local fourier spectrum analysis. In *Proceedings of the 14th Conference on Computer Graphics and Imaging (CGIM'13), Innsbruck, AUT*, volume 10, pages 2013–797. Citeseer, 2013.

[55] Jingning Shi, Wei Xiang, Qijing Liu, and Sher Shah. Mtreering: An r package with graphical user interface for automatic measurement of tree ring widths using image processing techniques. *Dendrochronologia*, 58:125644, dec 2019.

[56] P.T. Soule', P.A. Knapp, J.T. Maxwell, , and T.J. Mitchell. A comparison of the climate response of longleaf pine (pinus palustris mill.) trees among standardized measures of earlywood, latewood, adjusted latewood, and totalwood radial growth. *Trees*, 35, 2021.

[57] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[58] Jonathan von Oppen, Jakob J. Assmann, Anne D. Bjorkman, Urs A. Treier, Bo Elberling, Jacob Nabe-Nielsen, and Signe Normand. Cross-scale regulation of seasonal microclimate by vegetation and snow in the arctic tundra. *Global Change Biology*, 28(24):7296–7312, sep 2022.

[59] Kentaro Wada. Labelme: Image polygonal annotation with python. Zenodo, 2024.

[60] Fanyou Wu, Yunmei Huang, Bedrich Benes, Charles C. Warner, and Rado Gazo. Automated tree ring detection of common indiana hardwood species through deep learning: Introducing a new dataset of annotated images. *Information Processing in Agriculture*, 11(4):552–558, 2024.

[61] Hong Zhou, Rong Feng, Hua hong Huang, Er pei Lin, and Jun lin Yu. Method of tree-ring image analysis for dendrochronology. *Optical Engineering*, 51(7):077202, 2012.

Glossary

- **Annual growth ring**: A visible ring representing one year of growth, typically consisting of two zones: earlywood and latewood.
- **Area**: The two-dimensional surface enclosed by a ring boundary, usually measured in square millimeters (mm^2). It is known as well as Basal Area Increment (BAI).
- **Core sample**: A rectangular portion extracted from the disc, often used in dendrochronology because it allows growth analysis without felling the tree.
- **Crack**: A fracture or split in the disc, often caused by drying processes or mechanical stress.
- **Disc**: A cross-sectional image of a tree or shrub obtained by cutting perpendicular to the longitudinal axis of the stem or trunk.
- **Earlywood**: The lighter-colored portion of an annual ring, formed during spring and summer when growth is faster and cells are larger and thinner-walled.
- **Fungal stain**: Discoloration in the wood caused by fungal infection, which can affect the visibility of ring boundaries.
- **High pith eccentricity**: A structural characteristic in which the pith is located far from the geometric center of the stem or disc.
- **Knot**: A circular or irregular feature in the wood caused by the base of a branch embedded in the trunk.
- **Latewood**: The darker-colored portion of an annual ring, formed during autumn and winter when growth is slower and cells are smaller and thicker-walled.
- **Perimeter**: The total length of a ring boundary, used to quantify the shape and extent of a growth ring.
- **Pith**: The central part of the disc, corresponding to the first year of growth. Typically located near the geometric center. It is also known as the medulla.
- **Reaction wood**: A type of wood formed in response to mechanical stress (e.g., gravity or wind), typically seen as asymmetric growth. In conifers, it is called compression wood and forms on the lower side of leaning stems; in hardwoods, it is called tension wood and forms on the upper side.
- **Ring width**: The radial distance between two consecutive annual rings, often used as an indicator of yearly growth conditions.

Esta página ha sido intencionalmente dejada en blanco.

List of Tables

List of Figures

Esta es la última página.
Compilado el Wednesday 10th September, 2025.
http://iie.fing.edu.uy/