ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Development of an Intelligent and Efficient System for Monitoring and Optimising Sailboat Performance

Henrique Manuel de Almeida e Silva dos Santos Teixeira

Dissertation submitted in partial fulfilment of the requirements for the Master's degree in Critical Computing Systems Engineering

Supervisor: Luís Miguel Moreira Lino Ferreira

Co-Supervisor: Tiago Carlos Caló Fonseca

Evaluation Committee:

President:

Luís Miguel Pinho, Instituto Superior de Engenharia do Porto

Members:

António Castro, Instituto Superior de Engenharia do Porto

Luís Lino Ferreira, Instituto Superior de Engenharia do Porto

Porto, September 29th, 2025

Abstract

This thesis introduces an AI-powered tool that improves the analysis of sailing performance by automatically detecting thin, high-visibility stripes on sails. It uses computer vision and deep learning to extract key aerodynamic parameters, such as camber, draft, and twist. These parameters are essential for understanding sail shape and enhancing performance. The motivation lies in reducing reliance on traditional manual estimation methods while ensuring efficient onboard processing with lightweight devices like a GoPro camera connected to a tablet. The research starts with a review of current computer vision and AI-based image processing techniques. It also includes a sailing-specific look at the structural and aerodynamic features of sails. Several AI methods - Feature Extraction, Line Detection, Object Detection and Recognition, and Image Segmentation - are compared in this context. The analysis finds that semantic segmentation is the best technique for the goals of this thesis. A further comparison of semantic segmentation models - SegFormer, DeepLab, SAM, and Fast-SCNN - evaluated their accuracy, efficiency, and use for real-time deployment. This review shows that SegFormer is the most effective method for detecting lines in high-resolution images of a sailboat's sail. The evaluation carried out in this thesis compares a traditional algorithm, developed in a previous thesis and reused here as a baseline, with an AI-based approach that uses the SegFormer model. This implementation relies on the SegFormer mit-b1 backbone, chosen for its balance between accuracy and efficiency. Mit-b2 and mit-b3 were also tested for segmentation quality and processing time comparisons. The evaluation used a dataset of 23 videos and measured how well both methods could reliably detect lines for extracting aerodynamic parameters. The results show a clear trade-off. The traditional method consistently produced faster processing times because it relies on lightweight operations optimised for CPU use. In contrast, the SegFormer model offered more accurate and reliable line segmentation but required more computational power. Among the tested backbones, SegFormer mit-b1 was the best choice, as mit-b2 and mit-b3 resulted in significantly longer processing times without substantial improvements in segmentation accuracy. In conclusion, the traditional algorithm is still beneficial when speed and limited resources are critical. However, the AI-based approach, especially with SegFormer mit-b1, stands out as a reliable and precise option when more computational resources are available. This work illustrates the potential to integrate AI-driven computer vision into sailing performance analysis, aiding in the accurate and automated extraction of aerodynamic parameters to enhance decision-making and performance improvement in sailing.

Keywords: Accuracy, AI-based, Camber, Computational cost, Computer Vision, Draft, Efficiency, GPU, Image processing techniques, Image Segmentation, Sail aerodynamics, Sail structure, SegFormer, Semantic segmentation, Transformer-based segmentation, and Twist.

Resumo

Esta dissertação apresenta uma ferramenta suportada por inteligência artificial que melhora a análise do desempenho na vela através da deteção automática de riscas finas nas velas de veleiros. O sistema recorre a técnicas de visão por computador e *Deep Learning* para extrair parâmetros aerodinâmicos essenciais, como o camber, o draft e o twist. Estes parâmetros são fundamentais para compreender a forma da vela e otimizar o seu desempenho. A motivação do trabalho reside na redução da dependência de métodos tradicionais de estimativa manual, assegurando simultaneamente um processamento eficiente a bordo com dispositivos leves, como uma câmara GoPro ligada a um tablet. A investigação inicia-se com uma revisão das técnicas atuais de visão por computador e processamento de imagem baseadas em IA, incluindo ainda uma análise específica das características estruturais e aerodinâmicas das velas. Neste contexto, são comparados vários métodos de IA — extração de características, deteção de linhas, deteção e reconhecimento de objetos e segmentação de imagem. A análise conclui que a segmentação semântica é a técnica mais adequada para os objetivos desta tese. Posteriormente, é realizada uma comparação entre diferentes modelos de segmentação semântica — *SegFormer*, *DeepLab*, *SAM* e *Fast-SCNN* — avaliando a sua precisão, eficiência e aplicabilidade em contextos de utilização em tempo real. Esta revisão mostra que o *SegFormer* é o método mais eficaz para a deteção de linhas em imagens de alta resolução das velas de um veleiro. A avaliação experimental realizada nesta dissertação compara um algoritmo tradicional, desenvolvido numa dissertação anterior e reutilizado aqui como linha de base, com uma abordagem baseada em IA suportada pelo modelo SegFormer. A implementação recorre ao *backbone SegFormer* MiT-b1, escolhido pelo equilíbrio entre precisão e eficiência. Foram ainda testados os *backbones* MiT-b2 e MiT-b3, apenas para efeitos comparativos em termos de qualidade de segmentação e tempo de processamento. A avaliação utilizou um conjunto de 23 vídeos, medindo a capacidade de ambos os métodos em detetar linhas de forma fiável para a extração de parâmetros aerodinâmicos. Os resultados evidenciam um *trade-off* significativo. O método tradicional apresentou tempos de processamento consistentemente mais rápidos, uma vez que assenta em operações leves otimizadas para execução em CPU. Em contraste, o modelo *SegFormer* forneceu segmentações de linhas mais precisas e fiáveis, embora exigisse maior capacidade computacional. Entre os *backbones* testados, o *SegFormer* MiT-b1 revelou-se a melhor escolha, dado que o MiT-b2 e o MiT-b3 resultaram em tempos de processamento significativamente mais elevados sem melhorias relevantes na precisão da segmentação. Em conclusão, o algoritmo tradicional continua a ser vantajoso em cenários onde a velocidade e os recursos limitados são fatores críticos. Contudo, a abordagem baseada em IA, especialmente com o *SegFormer* MiT-b1, destaca-se como uma solução fiável e precisa quando estão disponíveis mais recursos computacionais. Este trabalho demonstra o potencial da integração da visão por computador suportada por IA na análise do

desempenho da vela, permitindo a extração automática e precisa de parâmetros aerodinâmicos que apoiam a tomada de decisão e a melhoria do desempenho em contextos de navegação à vela.

Palavras-chave: Aerodinâmica da vela, Algoritmos de IA, Arqueamento (camber), Custo computacional, Eficiência, Estrutura da vela, GPU, MiT-B1, Precisão, Profundidade (draft), Segmentação de Imagem, Segmentação Semântica, SegFormer, Técnicas de processamento de imagem, Torção (twist), Transformadores, Visão por computador.

Table of Contents

List of Figures

List of Tables

List of Codes

List of Equations

Acronyms and Symbols

List of Acronyms

AI	Artificial Intelligence
ASPP	Atrous Spatial Pyramid Pooling
CNN	Convolutional Neural Network
CRF	Fully Connected Conditional Random Fields
CV	Computer Vision
Fast-SCNN	Fast Semantic Segmentation Convolutional Neural Network
GPU	Graphics Processing Unit
IoU	Intersection over Union
ISEP	Instituto Superior de Engenharia do Porto
MAE	Mean Absolute Error
MiT	Mix Transformer
Mix-FFN	Mix-Feed Forward Network
MLP	Multi-Layer Perceptrons
OpenCV	Open-Source Computer Vision Library
RMSE	Root Mean Squared Error
SAM	Segment Anything Model
UI	User Interface
ViT	Vision Transformers

1 Introduction

The shape and trim of a sailing boat's sails are fundamental factors of its overall performance. Adjusting the sail shape requires balancing multiple parameters, including sail tension, mast angle, and curvature along the sail's surface. These adjustments influence how the wind interacts with the sail, affecting the boat's speed, stability and manoeuvrability. However, identifying the optimal sail configuration for a given set of conditions remains a complex and often imprecise process. Traditionally, sailors rely on experience, intuition, and real-time observations to fine-tune the sails, seeking an optimal performance based on the boat's response and speed.

Recent advancements in Artificial Intelligence (AI), Computer Vision (CV), and sensor technology offer a transformative approach to this process, enabling objective, data-driven analyses that complement the sailor's judgement. This thesis aims to modernise sail optimisation by integrating state-of-the-art technologies capable of analysing the shape of a sail and enabling the provision of real-time feedback to the sailor. The proposed system incorporates an AI-based algorithm designed to analyse the curvature of the sail, which a camera has acquired. By precisely interpreting the sail shape, the system can help determine the optimal sail configuration, enhancing performance in highly competitive sailing environments.

A key focus of the project is its practical implementation, emphasising the development of a lightweight and efficient solution that can operate seamlessly on portable devices such as GoPro cameras and tablets without excessive power consumption. This aspect is particularly critical for sailing applications, where compactness and energy efficiency are essential due to the limited onboard resources available. As a result, this system serves as a versatile tool, offering a comprehensive assessment of sailing performance. Ultimately, the project seeks to bridge the gap between traditional, experience-based sail tuning and a more precise, technology-assisted approach, setting a new benchmark for performance optimisation in sailing.

Figure 1 shows an example of a sailboat.

Figure 1 - Sailing Sport [1].

1.1 Background

The SoftCPS research group at the Instituto Superior de Engenharia do Porto (ISEP) actively collaborates with industry partners to advance research and innovation in developing cutting-edge technological solutions. This initiative builds upon the foundations of the SailShape project, a collaborative effort between ISEP and SVDR Lda, a company specialised in intelligent systems for sails and drones. As an extension of SailShape, this project seeks to design and implement an AI-driven algorithm capable of more efficiently and accurately identifying lines in a sail, enabling the precise calculation of parameters for optimising sail performance.

1.2 Motivation and Context

Sailing is a dynamic and complex activity, where success is determined upon a delicate balance of skill, knowledge, and adaptability to changing environmental conditions. The optimisation of a boat's sail to achieve peak performance is a delicate task, influenced by multiple factors such as wind speed, wave patterns, boat velocity, and the sailor's expertise. Traditionally, sailors rely on empirical methods, adjusting their sail configurations over time based on feedback from competitive comparisons and race outcomes. However, this approach is often slow, inconsistent, and susceptible to personal judgments. Even minor sail tuning refinements in highly competitive sailing scenarios can significantly improve speed and manoeuvrability, highlighting the potential benefits of a more precise, data-driven methodology.

The motivation for this project was derived from the increasing interest in utilising technology to enhance sports performance while reducing reliance on traditional manual techniques. Recent advancements in artificial intelligence (AI) have made it possible to

analyse sail shapes using CV, enabling accurate detection of sail lines and facilitating a more precise determination of airflow angles. Unlike human observation, which is inherently limited, an AI-based system provides objective, real-time feedback on sail shape and orientation. Furthermore, this system continuously monitors and analyses critical parameters such as camber, draft, twist, entry and exit angles — factors that are challenging to assess with precision through conventional manual observation.

Additionally, energy efficiency is a fundamental consideration in this project, given that sailing activities often extend over prolonged durations without access to convenient power sources. The system is designed to employ low-power algorithms and efficient data processing techniques to ensure functionality under limited battery power. Simultaneously, the system highlights high performance, delivering accurate and timely data analysis without sacrificing speed or reliability. This dual emphasis on energy efficiency and performance is particularly crucial for portability, as the system is intended to operate primarily on devices such as GoPro cameras and tablets, which are widely used in sailing due to their compactness and versatility. The system's ability to collect and analyse data efficiently makes it a practical and reliable tool for sailors, providing valuable insights without imposing excessive power demands.

Aside from these inspirations, this work is also derived from a previous project — SailShape — which was developed with traditional image processing techniques. In their previous work, the project demonstrated the detectability of sail lines with thresholding, contour detection, and calibration procedures. However, while robust, these demonstrated limitations in providing accuracy under variable lighting conditions. From this background, this current work aims to take the methodology further by incorporating AI-based algorithms to determine whether they can outperform traditional approaches in both detection accuracy and computational efficiency. This continuity asserts that the present work is no innovation, but indeed a continuation of previous work, and provides comparative insights between traditional and AI-based image analysis for sail optimisation.

1.3 Goal and Objectives

The primary goal of this work is to identify critical parameters related to the sail shape of a sailing boat. To achieve this goal, the project focuses on the following specific objectives:

1. Traditional image processing techniques and computer vision methods will be studied deeply. The state-of-the-art review would contrast and compare traditional and AI-based image processing techniques in terms of their stability while working with images of large resolutions, varying illumination, and their balance between performance and computational complexity.
2. The work that is being proposed will create and calibrate a sail line detecting algorithm that can rapidly and accurately identify sail lines. It must work efficiently regardless of sail conditions (i.e., changing light levels, shadows, and wave

conditions) and continuously produce proper measurements of parameters such as camber, draft, and twist. It would be evaluated with benchmark datasets and traditional image processing techniques to determine the processing speed and accuracy.

3. Power efficiency is a top requirement as it would be run from portable gear such as action cameras and tablets. The algorithm would be optimised to minimise power consumption without being unresponsive to be displayed in real-time. Model compression, memory-efficient coding, and judicious processing of frames would be considered to enhance battery longevity while stretching sail sessions over hours.

While the immediate focus is on achieving reliable, efficient, and accurate sail line detection, the following objectives are defined as longer-term extensions of the project:

4. Incorporate additional sensors for real-time data acquisition, including 3-axis accelerometers, gyroscopes, and magnetometers.
5. Build a structured dataset encompassing diverse sailing conditions, enabling further analysis and insights.
6. Utilise collected data to detect wave patterns, assess boat movement (e.g., ascending or descending a wave), and provide actionable recommendations to the sailor.
7. Expand the system's compatibility to support additional camera models and enable seamless integration with existing navigation systems on the boat (optional).

Figure 2 illustrates the expected results for the AI-based algorithm with the measurements of the necessary sailing parameters.

Figure 2 - Expected Results [2].

1.4 Methodology and Timeline

The development of this project follows a structured methodology to ensure a systematic and efficient workflow. The process is divided into several key phases, each contributing to the successful implementation of the system:

- **Problem and Motivation (02 December 2024 – 14 December 2024):** This phase identifies the problem the project aims to solve and highlights its significance. It establishes the need for a reliable solution and explains the motivation behind the study within its research context.
- **Goals and Objectives (10 December 2024 – 14 December 2024):** Clearly define the project's main goals and specific objectives that must be achieved. These objectives serve as measurable milestones to effectively guide the research and development process.
- **State of the Art and existing approaches research (16 December 2024 – 20 January 2025):** Review and analyse existing methods, techniques, and technologies relevant to the project. This research helps identify the best approach while understanding the strengths and limitations of current solutions.
- **Analysis and Design of the System (20 January 2025 – 05 February 2025):** Focuses on defining the system's architecture and selecting the appropriate frameworks and tools. This stage also includes developing a prototype to validate the proposed approach before full implementation.
- **Development of the System (05 February 2025 – 8 July 2025):** Covers the actual construction of the system, including software development, model training, and component integration. The implementation is carried out according to the design specifications to ensure the system meets the intended objectives. During this stage, a range of alternative algorithms and implementations will be evaluated to determine the most appropriate solution for the system.
- **Testing, Evaluation, and Experimental Validation (8 July 2025 – 18 August 2025):** This phase involves rigorous testing to assess the system's performance, reliability, and accuracy. Experimental validation is conducted to verify that the implemented solution meets the initial goals, with refinements applied as needed.
- **Writing of the Dissertation (16 December 2024 – 25 September 2025):** Runs parallel with the other phases, documenting the research, design, implementation, and results. The dissertation writing ensures that the project findings are systematically recorded for final submission.

The following work plan in Figure 3 will guide the methodology outlined previously.

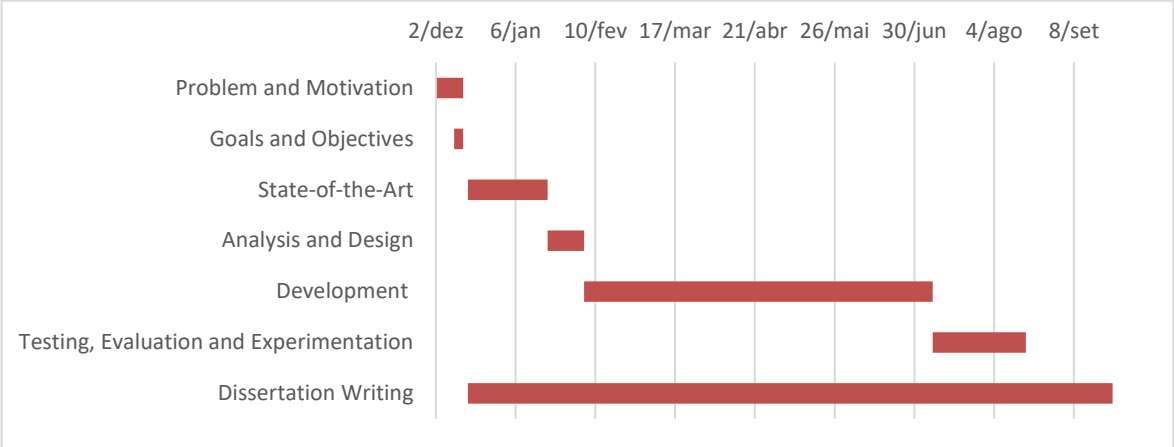


Figure 3 - Project Timeline Gantt Chart.

1.5 Thesis Structure

This thesis is organised into six main chapters, each addressing different aspects of the research and development process. The structure ensures a logical progression from the problem context to the concepts related and chosen solutions. Below is the description of the thesis's structure:

- **Chapter 2, Background in Sailing:** Introduces fundamental concepts related to sailing, particularly focusing on sails' structural and aerodynamic characteristics. It discusses critical parameters such as camber, draft, and twist, essential for understanding the problem domain.
- **Chapter 3, State of the Art:** Provides an in-depth review of existing methods and technologies in CV and AI-based image processing techniques. It explores different approaches, including feature extraction, line detection, object detection, and image segmentation. Additionally, it compares various AI-based semantic segmentation models, such as SegFormer, DeepLab, Segment Anything Model, and Fast Semantic Segmentation Convolutional Neural Networks, leading to selecting the most suitable approach for the research.
- **Chapter 4, System Architecture:** This chapter outlines the overall architecture of the proposed solution. It describes the workflows for both model training and inference, explaining how data moves through the system and how sail parameters are calculated.
- **Chapter 5, System Implementation:** This chapter focuses on how the proposed approach is implemented. It discusses preparing datasets, strategies for augmenting data, loss functions to address class imbalance, training methods, and evaluation metrics. It also describes the inference pipeline, which includes extracting video

frames, processing segmentation, post-processing steps, and calculating sail parameters.

- **Chapter 6, Comparative Evaluation:** This chapter provides an experimental evaluation of the proposed approach. It describes the hardware setup, the method for comparison, and processing time analysis, and it evaluates the quality of segmentation and the accuracy of parameter computation.
- **Chapter 7, Conclusions:** This chapter summarises the research findings, highlighting contributions, limitations, and achieved goals. It also suggests ways for future work, detailing how the system could be further improved or expanded in later research.

2 Structural and Aerodynamic Features of Sails

This section provides an overview of the fundamental principles of sail design and their impact on sailing performance, with an emphasis on the structural and aerodynamic features most relevant to this project. Given that the system aims to detect sail lines and extract aerodynamic parameters such as camber, draft, and twist, a solid understanding of these characteristics is essential. The discussion highlights how sail geometry and airflow dynamics form the foundation for applying AI-based techniques. Accurate sail line detection plays a crucial role, as it defines key reference lines used to quantify sail shape, optimise trim, and assess performance in real-world sailing conditions. The literature review of this chapter is presented in the Annexe.

2.1 Structural Features of the Sail

The sail of a sailboat is a complex structure, with several key features that significantly influence its shape, aerodynamic performance, and overall functionality. As illustrated in Figure 4, understanding these features is fundamental to this project, which focuses on detecting sail lines, analysing wave patterns, and optimising sailing performance. The following sections provide an overview of the most critical elements [2], [3]:

1. **Head**: The uppermost corner of the sail, where the leech and luff converge. The head is crucial in determining the total height of the sail, which directly influences aerodynamic efficiency and the power generated.

2. **Leech**: The trailing edge of the sail, extending from the head to the clew. The shape and tension of the leech play a significant role in governing airflow over the sail, thereby impacting performance. Accurate identification of the leech is essential for sail trim analysis and detecting areas where adjustments are needed.

3. **Luff**: The sail's leading edge runs from head to tack. The luff is typically attached to the mast (in mainsails) or the forestay (in jibs) and is essential in defining the sail's angle relative to the wind. Precise detection of the luff allows for a more accurate evaluation of the sail's alignment with wind direction.

4. **Foot**: The bottom edge of the sail, connecting the tack to the clew. The foot establishes the base of the sail and determines its interaction with the deck. Proper identification of the foot is vital for calculating the total sail area.

5. **Clew**: The lower aft corner of the sail, where the leech and foot intersect. The clew's position is a crucial reference point in determining the sail's geometry and trim, as it anchors the bottom trailing edge of the sail.

6. **Tack**: The lower forward corner of the sail, where the luff and foot meet. The tack is a fixed point that defines the base alignment of the sail with either the mast or the forestay. Identifying this point is essential for establishing a stable reference for other sail parameters.

7. **Batten**: Horizontal or diagonal reinforcements are inserted into pockets along the sail to maintain structural integrity and prevent excessive deformation. Detecting battens accurately aids in understanding the sail's curvature and flexibility, which influence aerodynamic performance.

8. **Boom Line**: An imaginary line extending along the boom (the horizontal spar supporting the bottom of the sail), running from the front (mast side) mast to the back (aft end). The boom line represents the range of motion the boom follows as it adjusts the sail in response to wind conditions.

9. **Chord Line**: A straight line connecting the leading edge (luff) to the sail's trailing edge (leech). It is an idealised representation of the sail's profile, cutting through its natural curvature. The angle between the chord line and the wind is critical to the sail's aerodynamic efficiency.

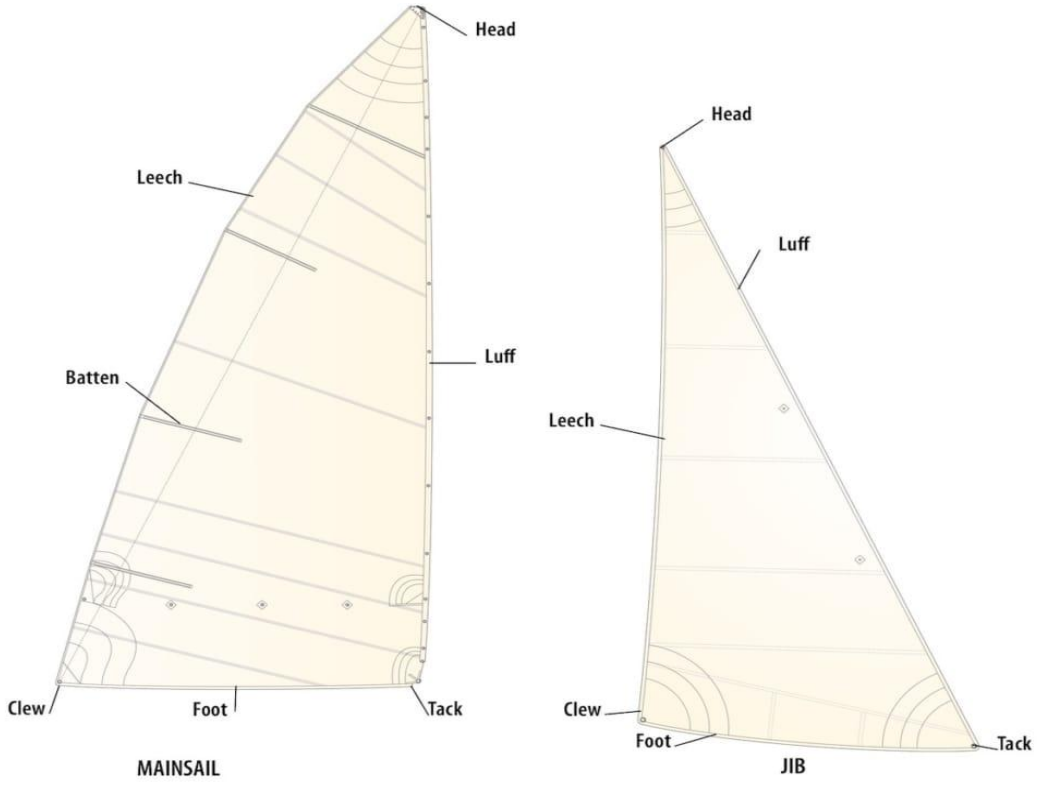


Figure 4 - Structural Features of the Sail [3].

The AI-based algorithm will utilise the luff and leech as the foundation for extracting and calculating critical aerodynamic parameters, including camber, twist, and draft (as detailed in Chapter 2.2). These parameters play a fundamental role in evaluating the sail's aerodynamic efficiency.

2.2 Aerodynamic Features of the Sail

The shape and structure of a sail determine how effectively it interacts with the wind to generate propulsion. Several critical parameters — **Camber, Draft, Twist, Entry Angle** and **Exit Angle** — define the aerodynamic profile of the sail and its adaptability to varying wind and sea conditions. These parameters comprehensively represent the sail's geometry, directly influencing the boat's speed, manoeuvrability, and stability. Figure 5 outlines these essential characteristics:

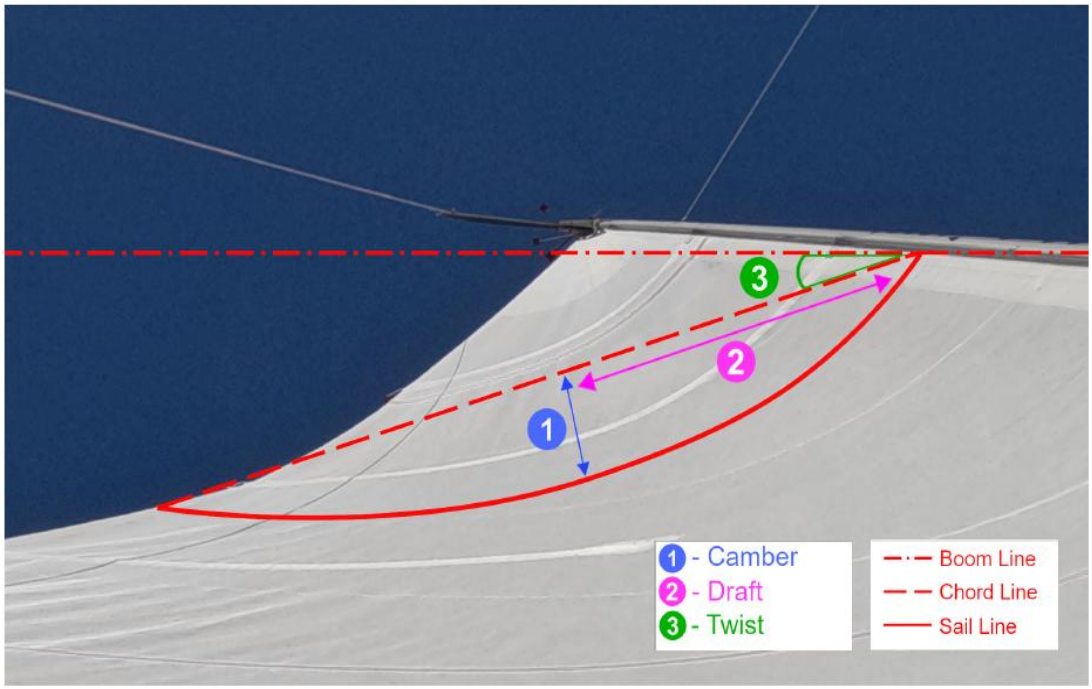


Figure 5 - Camber, Draft and Twist [2].

2.2.1 Camber

Camber refers to the curvature or fullness of the sail. A sail with a higher camber is fuller and generates greater power, which is particularly advantageous in light wind conditions, as it enhances lift and improves speed. Conversely, a flatter sail with a lower camber produces less lift and reduces drag, making it more effective in strong winds, where excessive power could destabilise the boat. The balance between camber and wind strength is crucial: greater camber is preferred in light winds to maximise propulsion, while a reduced camber is

necessary in heavier winds to maintain stability and control [2], [4], [5]. Below is Equation 1, which calculates the camber percentage.

$$Camber\ \% = \frac{Camber \times 100}{Chord\ Line} \quad \text{Eq. (1)}$$

Equation 1 - Camber Calculation.

2.2.2 Draft

The Draft represents the depth of the sail's curvature. A deeper draft increases lift, which is beneficial in light wind conditions where additional power is needed to propel the boat. However, a flatter sail with less draft is preferred in stronger winds, as it reduces the amount of power generated, preventing excessive heeling and maintaining stability. Additionally, the position of the draft — whether closer to the luff or the leech — is a crucial factor in performance optimisation. Adjusting the draft position enables a balance between power generation and boat stability, making it an essential tuning parameter for varying sailing conditions [2], [4], [6]. Equation 2 demonstrates how to calculate the draft's percentage.

$$Draft\ \% = \frac{Draft \times 100}{Chord\ Line} \quad \text{Eq. (2)}$$

Equation 2 - Draft Calculation.

2.2.3 Twist

Twist refers to the variation in the angle of the sail's chord line from the bottom of the sail (near the foot and clew) to the top (near the head). The lower part of the sail is trimmed to a specific angle, while the upper portion may twist, resulting in a different exposure to the wind. This occurs because wind speeds vary at various heights, with slower winds near the water's surface due to friction. In light wind conditions, allowing more twist at the top of the sail can be beneficial, as it enables the sail to capture stronger winds at higher altitudes while reducing drag at the bottom, ultimately enhancing speed. In contrast, in strong winds, minimising twist helps maintain control by keeping the sail better aligned, preventing excessive heeling or overpowering the boat [2], [4], [7]. Equation 3 represents the equation for the calculation of Twist.

$$Twist\ º = tan^{-1}(Chord\ Line) \times \frac{180}{\pi} \quad \text{Eq. (3)}$$

Equation 3 - Twist Calculation.

2.2.4 Maximum Camber Point

The maximum camber point is the location on the sail curve that lies the farthest away from the straight chord line between the line endpoints. In other words, it is where the distance between the curve of the line of the sailboat $f(x)$ and the chord $y_{chord}(x)$ reaches its maximum:

$$x_c = \arg max_{x \in [x_1, x_2]} |f(x) - y_{chord}(x)| \qquad \text{Eq. (4)}$$

$$C_{max} = (x_c, f(x_c))$$

Equation 4 - Calculation of Maximum Camber Point.

3 State of the Art

This section provides a comprehensive overview of key advancements in CV and AI-based image processing techniques, outlining fundamental concepts essential for visual data analysis. It first explores core methodologies, such as feature extraction, line detection, object detection & recognition, and image segmentation, emphasising their role in image processing and interpretation. A comparative analysis of advanced image processing techniques is conducted to determine the most suitable approach for the project. Based on this selection, specific AI-based models are examined in detail. Finally, a comparative analysis of these AI-based models is performed to evaluate their strengths and weaknesses, ensuring the selection of the most effective solution for the given application. The literature review of this chapter is outlined in the Annexe.

3.1 Computer Vision Main Concepts

Computer vision is a subfield of artificial intelligence (AI) that enables machines to interpret and analyse visual data from the real world. By simulating human vision, this technology allows computers to process images, recognise patterns, and extract meaningful information [8], [9]. In sailing, CV offers an innovative approach to analysing sail performance by detecting specific visual markers — such as orange lines placed on the sail — to determine key aerodynamic parameters. This method provides deeper insights into sailing behaviour under varying wind conditions, ultimately contributing to the development of more efficient sail designs and enhanced sailing performance [10], [11].

CV integrates image processing, pattern recognition, and machine learning techniques as a multidisciplinary field to extract valuable insights from visual data. The primary goal of CV is to enable computers to "see" and interpret images or video sequences in a manner that facilitates decision-making or further computational analysis. This technology has been successfully applied across various domains, including facial recognition [12], medical imaging [13], autonomous vehicles [14] and industrial automation [15].

In sailing applications, CV techniques are employed to identify markers on sails, analyse their movement, and extract essential performance parameters such as camber, draft, and twist. By leveraging these advancements, sailors and researchers can better understand sail dynamics, leading to optimised performance and improved sailing efficiency.

At its core, CV relies on:

- **Digital Image Processing:** The manipulation and enhancement of images to improve interpretation and analysis [16].

- **Pattern Recognition:** The identification of objects, shapes, and textures within an image [16], [17], [18].
- **Feature Detection:** The process of locating specific markers or patterns in an image to extract relevant information [19].
- **Machine Learning and AI Models:** The application of trained algorithms to recognise and classify objects based on previously acquired data [16].

The integration of CV in sailing offers several notable advantages. One of the primary benefits is its **enhanced accuracy** [20], as advanced image processing and recognition techniques enable the precise detection and measurement of sail parameters. This ensures a more reliable assessment of key aerodynamic properties, such as camber, draft, and twist, ultimately leading to improved performance analysis.

Another significant advantage is **real-time monitoring** [21], which allows for continuous tracking of sail behaviour without requiring manual observations. The ability to collect and process real-time data facilitates **informed decision-making** [22], [23], as sailors and designers receive immediate feedback on sail adjustments. This instant feedback loop enables dynamic optimisation of sail configurations, improving overall performance.

Furthermore, the data acquired through CV systems plays a crucial role in **sail design optimisation**, as data-driven insights, researchers and designers can refine sail structures and materials for future applications. This iterative approach enhances efficiency and performance in both competitive and recreational sailing.

3.2 AI-Based Image Processing Techniques

This section examines AI-based image processing techniques, focusing on four key methodologies: **Feature Extraction, Line Detection, Image Segmentation, and Object Detection & Recognition**. These techniques are the foundation for modern CV applications, enabling AI systems to analyse, interpret, and make informed decisions based on visual data. Each method plays a distinct role, and selecting the most suitable approach depends on the project's specific objectives and considerations, such as accuracy, computational efficiency, and real-time performance.

3.2.1 Feature Extraction

Feature extraction is a fundamental process in CV and machine learning that acts as a bridge between raw data and intelligent decision-making [24]. It involves transforming high-dimensional data — such as images or complex datasets — into a more compact and meaningful representation [25]. This transformation enables AI systems to analyse, classify, and interpret data efficiently while reducing computational complexity, enhancing accuracy, and improving the robustness of predictive models.

As a critical component of AI-driven applications, feature extraction facilitates precise and reliable data interpretation across various domains. This technique enhances the effectiveness of AI models, enabling their application in diverse fields requiring high levels of accuracy and efficiency:

- **Medical Imaging:** AI-driven feature extraction techniques facilitate the detection of anomalies such as tumours, fractures, and tissue abnormalities by analysing X-rays, MRIs, and CT scans. These methods enhance diagnostic accuracy and support early prognosis by highlighting critical details within medical images [26].
- **Autonomous Vehicles:** In self-driving technology, AI-based feature extraction enables real-time identification of lane markings, traffic signs, pedestrians, and obstacles. This process enhances object recognition and situational awareness, contributing to the safe navigation of autonomous vehicles [27].
- **Facial Recognition:** AI models utilise extracted facial features — such as the shape of the eyes, nose, and mouth — to distinguish between individuals with high precision. This capability is fundamental to security systems, authentication protocols, and social media applications [28].
- **Data Mining:** feature extraction transforms raw data into meaningful representations, allowing machine learning models to process, analyse, and extract valuable insights efficiently. This technique is widely used in customer analytics, financial fraud detection, text mining, and recommendation systems, significantly improving the accuracy and efficiency of data-driven applications [29].

Despite its importance, feature extraction presents several challenges that must be addressed to ensure optimal performance in AI systems:

- **Variability in Data**: Real-world datasets, particularly images, are subject to variations in lighting conditions, occlusions, distortions, and noise, impacting the quality and reliability of extracted features.
- **Computational Efficiency**: Ensuring that extracted features are both computationally efficient and informative enough to improve learning accuracy remains a critical research challenge.
- **Interpretability**: While deep learning-based feature extraction has significantly advanced AI capabilities, the resulting features are often complex and challenging to interpret. This lack of transparency raises concerns about the explainability and trustworthiness of AI models.

Feature extraction is fundamental across diverse domains, including CV, medical diagnostics, autonomous systems, and text processing. The ability to accurately extract and structure relevant features directly influences the performance of machine learning models. While traditional methods laid the foundation for this field, advancements in deep learning have automated and enhanced feature extraction, making AI systems more powerful, adaptable, and efficient.

Figure 6 demonstrates the feature extraction process in the context of image processing, which is a fundamental step in the analysis and interpretation of visual data. The process begins with a complete motorcycle image, representing the raw visual data. Feature extraction aims to identify and isolate smaller, significant components within the image that can effectively represent the object. This is accomplished through a feature extraction algorithm, which evaluates the input image to detect essential patterns or structures, such as edges, shapes, or textures.

The right-hand side of the figure presents the outcome of this process—a grid of extracted features. Each feature corresponds to a motorcycle element, including components like the wheels, bodywork, or suspension. These features capture critical details about the object while discarding irrelevant information, enabling the system to handle the image data more efficiently.

The primary objective of feature extraction is to reduce the complexity of the data, making it suitable for tasks such as object detection or recognition. The algorithm simplifies the image into key elements by concentrating on the object's most distinguishing components. This approach facilitates accurate analysis while handling variations in factors such as lighting, perspective, or partial occlusions.

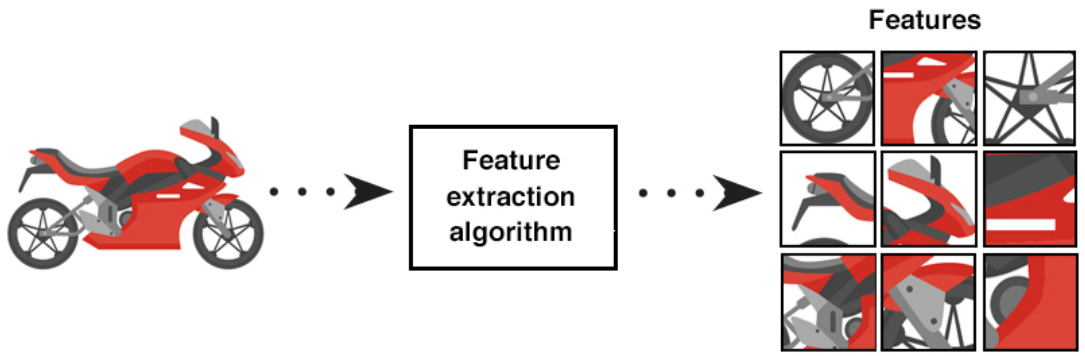


Figure 6 - Feature Extraction [25].

3.2.2 Line Detection

Line detection is a fundamental technique in CV that involves identifying straight or curved lines within an image or video frame. This technique plays a crucial role in analysing visual data's structure, geometry, and patterns. In many applications, lines represent edges, boundaries, or directional paths, making their detection essential for tasks such as object recognition, navigation, and scene understanding [30], [31].

Therefore, Line Detection is essential for tracking the sailboat's parameters in an intelligent system. The foremost task is to detect sail lines positively to help in knowing the orientation and position of the sail concerning the wind. It also includes recognising the sail's edges and outlines, which are easily affected by light, motion, and the atmosphere. The system must

efficiently process images or video feeds in real-time to capture and analyse these lines, enabling the system to adjust and optimise the sail's performance [32].

In the context of intelligent sailing systems, line detection is particularly significant for tracking key sailboat parameters. The primary objective is to accurately identify sail lines to determine the orientation and position of the sail relative to the wind. Additionally, the detection of edges and outlines of the sail is critical, as these features are influenced by varying light conditions, motion, and atmospheric factors. To ensure adequate performance, the system must process images and video feeds efficiently in real-time, enabling continuous monitoring and optimisation of sail dynamics.

An AI-based algorithm is employed for this task, utilising deep learning techniques to reduce time and cost while enhancing accuracy. The algorithm employs convolutional neural networks (CNNs) [33], which are capable of recognising very complicated patterns and distinctive features of the sail to provide accurate identification of lines in adverse conditions [34]. This approach handles various complexities such as dynamic backgrounds, fluctuating lighting, and different sail configurations.

Ultimately, the line detection system aims to provide real-time feedback on the sail's position, improving navigational control and optimising sailing performance. Through AI-powered detection, the system can quickly identify critical parameters of the sail, enabling adjustments that enhance efficiency and overall sailboat operation.

Figure 7 illustrates an AI-driven approach to **AI-based line detection** in image processing, demonstrating how structural lines are extracted from images. This process is divided into three distinct stages, using two example scenes — a modern bedroom and a living room — to highlight the transformation.

In the first row, the original images of the scenes are overlaid with green lines that represent the detected structural elements, such as the edges of furniture, walls, windows, and other significant interior components. This visualisation shows how line detection algorithms identify geometric features within complex environments, concentrating on lines that define the room's spatial structure and essential elements.

The second row displays the original images without the overlays, serving as a reference point for comparison.

The third row presents the output of the line detection process in black and white, highlighting the detected edges. This stage transforms the images into simplified representations, reducing them to lines and contours. The focus is on extracting line boundaries that define architectural and object features, such as window frames, furniture edges, and floor patterns. These outputs are devoid of colour and texture, reducing the image to its structural essence.

This example illustrates how AI-based line detection algorithms identify and isolate meaningful geometric structures in images, making them useful for applications such as

architectural analysis, object detection, and scene understanding. The process simplifies the visual data while retaining critical information about the layout and components of the scene.

This example illustrates how AI-based line detection algorithms identify and isolate meaningful geometric structures in images, making them useful for applications such as architectural analysis, object detection, and scene understanding. The process simplifies the visual data efficiently while preserving critical structural information about the layout and components of the scene.

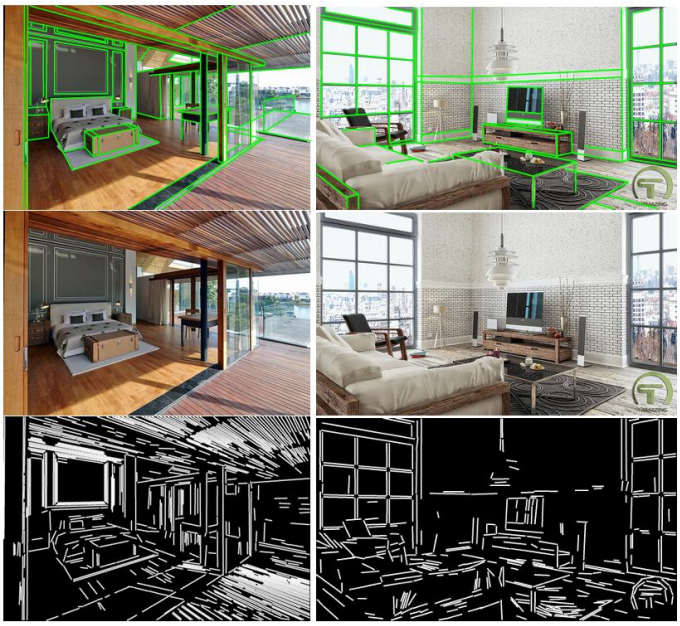


Figure 7 - Line Detection [35].

3.2.3 Object Detection and Recognition

Object detection and recognition have become fundamental components of CV, enabling machines to perceive, analyse, and interpret visual data with remarkable accuracy [36], [37]. The ability of AI systems to detect objects within an image, determine their positions, and recognise them as distinct entities is crucial in developing intelligent, autonomous technologies capable of operating in dynamic and complex environments.

Object detection involves identifying the presence of objects in an image or video and determining their location using bounding boxes or segmentation masks [37], [38], [39]. This process is essential for applications requiring spatial awareness, such as tracking moving objects in a video stream, detecting pedestrians in self-driving cars, or monitoring inventory in warehouses. On the other hand, object recognition classifies detected objects into predefined categories, enabling AI models to differentiate between various entities within an image [37], [39]. Together, these capabilities form the foundation of modern CV

applications, providing machines with the perceptual intelligence necessary for meaningful interaction with their surroundings.

Object detection and recognition have a broad range of applications across multiple industries and technological domains, including:

- **Autonomous vehicles:** AI-driven detection systems identify pedestrians, traffic signals, road signs, and other vehicles in real-time, enhancing navigation safety and accident prevention [27].
- **Security and Surveillance:** Object recognition plays a critical role in facial recognition systems, anomaly detection, and crowd monitoring, enhancing safety and security measures in public spaces [40], [41].
- **Medical Imaging:** AI-powered object detection systems facilitate the automatic identification of abnormalities such as tumours, fractures, and lesions in medical imaging scans (e.g., X-rays, MRIs), significantly improving diagnostic accuracy and enabling early disease detection [42].
- **Retail and E-commerce:** AI-based Object recognition enhances self-checkout systems, inventory management, and product recommendation engines, revolutionising the shopping experience [43].
- **Robotics and Automation:** Robots equipped with object detection capabilities can interact with objects, navigate environments, and perform complex tasks with greater precision and efficiency [44].

Advances in deep learning, machine learning, and computational power have fuelled the widespread adoption of AI-based object detection and recognition. Traditional methods relied on manually engineered features and handcrafted rules to detect and recognise objects. However, these approaches often struggled with variations in lighting, occlusions, and complex backgrounds, limiting their robustness [45]. The emergence of deep learning, particularly CNNs and transformer-based models, has revolutionised the field by enabling models to learn feature representations directly from data. Modern AI-based detection systems now demonstrate superior generalisation capabilities across diverse environments, achieving unprecedented levels of accuracy and efficiency in object identification tasks.

As object detection and recognition technologies evolve, their potential applications will expand, shaping the future of intelligent automation, smart cities, and AI-powered decision-making. These innovations will further enhance the capabilities of autonomous systems, improve efficiency across industries, and contribute to the ongoing transformation of AI-driven solutions in real-world applications. Figure 8 demonstrates an example of **object detection and recognition** in image processing, illustrating the capabilities of AI-based techniques to identify and label objects within a visual scene.

Within the image, various objects are outlined by bounding boxes, each labelled with a specific category such as "Human," "Dress," or "Purse." These bounding boxes represent the

output of an AI-based detection algorithm that processes the image to locate objects of interest and assigns them appropriate labels.

The algorithm performs the following steps:

1. **Object Detection**: The algorithm scans the image to detect regions of interest containing distinct objects, such as a human figure, clothing, and accessories.

2. **Feature Analysis**: Using advanced AI models (such as CNN), the algorithm analyses features like shapes, textures, and patterns to differentiate one object from another.

3. **Classification**: Once objects are detected, the algorithm categorises them into predefined categories, assigning labels based on their detected characteristics.

For example, the labelled bounding boxes in the figure below demonstrate the algorithm's ability to accurately identify/recognise objects, even in complex scenes containing multiple and overlapping elements.

- The "Human" box encircles the figure of a woman holding shopping bags, correctly identifying her as the subject.
- The "Dress" box isolates the mannequin wearing a red dress, identifying it as a separate object.
- The "Purse" box highlights a smaller item, showing the system's precision in detecting objects of varying sizes.

This example shows how AI-based techniques detect objects and separate them into meaningful categories. Although it deviates from line detection, it highlights how AI models leverage edge and feature detection as part of their processing pipeline to analyse and understand scenes. This approach is fundamental for automated shopping assistants, intelligent surveillance systems, and autonomous navigation.

This example underscores how AI-based techniques detect objects and separate them into meaningful categories.

Figure 8 - Object Detection and Recognition [46].

3.2.4 Image Segmentation

Segmentation is a fundamental technique in CV that involves partitioning an image into distinct regions, where each region consists of pixels that share specific attributes such as colour, texture, or semantic meaning [47], [48]. Unlike classification tasks, which assign a single label to an entire image (e.g., determining whether an image contains a cat or not), segmentation enables a more detailed analysis by allowing for pixel-level categorisation [48], [49]. This higher level of precision is essential in various real-world applications, including specialised tasks such as detecting specific features on boat sails, where fine-grained image analysis is required for accurate assessment and optimisation.

In the context of sailboat performance analysis, detecting orange lines on a sail requires a segmentation algorithm capable of distinguishing thin, elongated structures from the surrounding sail fabric. Given that these lines may vary in thickness, orientation, and visibility due to factors such as sail folds, wrinkles, and changing lighting conditions on the water, a robust segmentation approach must effectively account for subtle variations in colour and texture while preserving fine boundary details.

An AI-based solution designed to identify each orange line can benefit from methodologies inspired by Segmentation. By leveraging pixel-level analysis, the algorithm ensures precise localisation of every orange line on the sail. This capability facilitates further study, including measuring line lengths, detecting signs of wear and tear, and integrating the segmentation results into higher-level applications, such as real-time monitoring during sailing competitions.

Beyond mere detection, accurately segmenting these lines enables automated quality control, such as assessing sail line straightness and structural integrity and facilitating comprehensive structural evaluations. Given the requirement for per-pixel precision under variable maritime conditions, advanced segmentation techniques that integrate both local texture information and broader contextual awareness are essential [50]. Recent advancements incorporating global self-attention mechanisms [51] have the potential to further enhance segmentation accuracy by mitigating ambiguities in complex or visually cluttered environments.

Through these innovations, segmentation extends well beyond conventional image-level classification, offering a high degree of precision that supports a wide range of applications. In the context of sailboats, it provides the fine-grained, pixel-level detail necessary for accurately identifying each orange line, thereby enabling data-driven decision-making in sail maintenance, performance optimisation, and safety assessments.

Over time, segmentation research has evolved into multiple subcategories, each addressing distinct objectives and levels of specificity. Three prominent types — **semantic segmentation, instance segmentation, and panoptic segmentation** — are illustrated in Figure 9.

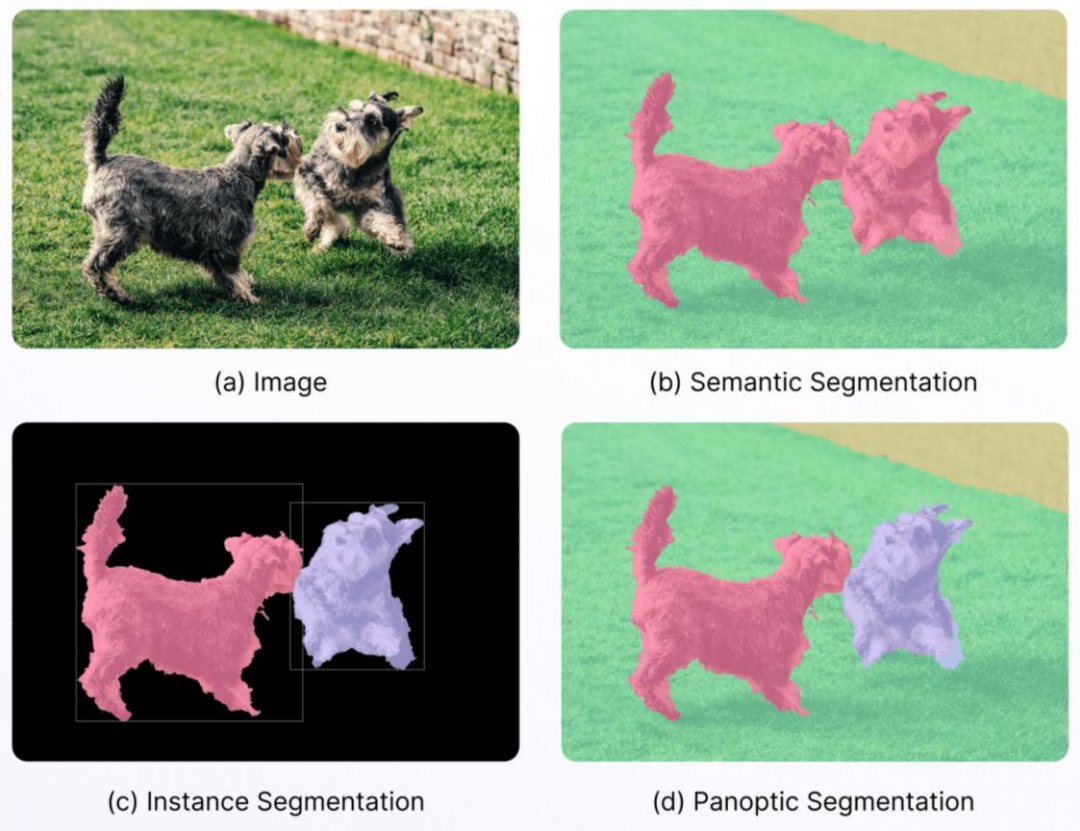


Figure 9 - Segmentation Techniques [52].

Semantic Segmentation

Semantic segmentation is a CV technique in which every pixel within an image is assigned to a predefined class (or category) [52]. This approach groups all objects of the same class without distinguishing between individual instances. For instance, in the example shown (Figure 9 (b)), the dogs are grouped as a single class (marked in pink), and the background, such as grass, is categorised as a different class (green). This approach groups all objects of the same type into one category without differentiating individual instances.

This technique is particularly relevant for applications requiring a broad understanding of scene composition. Semantic segmentation is essential for identifying roads, sidewalks, and pedestrians in autonomous driving to ensure safe navigation. Similarly, in medical imaging, delineating different tissue types using semantic segmentation aids in diagnosis and treatment planning. However, a key limitation of this approach is its inability to differentiate between multiple instances of the same class within an image.

Instance Segmentation

“Instance segmentation gives different labels for separate instances of objects belonging to the same class” [52], [53]. Unlike traditional segmentation methods that classify all objects of a particular type as a single entity, instance segmentation differentiates between multiple

occurrences of the same object. In Figure 9 (c), the two dogs are distinguished as individual entities, one in pink and the other in purple, with bounding boxes around them.

Significant advancements in instance segmentation include frameworks that integrate object detection and segmentation into a unified process [54], real-time systems optimised for balancing speed and accuracy [55], and methodologies that employ conditional filters to streamline per-instance predictions [56]. These innovations enhance the efficiency and precision of instance segmentation, making it a crucial tool for applications requiring detailed object-level analysis.

Separate instances of the same class are given unique labels.

Panoptic Segmentation

Panoptic segmentation combines semantic and instance segmentation principles to interpret an image comprehensively. In Figure 9 (d), the dogs are individually identified (as in instance segmentation), while the background, such as grass and walls, is grouped into continuous areas (as in semantic segmentation).

Panoptic segmentation ensures that all elements within an image are accurately labelled, preserving both spatial context and object identity, by integrating both segmentation techniques. Recent advancements in this field have focused on improving efficiency and refining segmentation processes to enhance performance in large-scale and complex scenes [52], [57].

3.2.5 Comparative Analysis and Selection of the Best Image Processing Technique

Semantic segmentation emerged as the most effective and reliable method among the various AI-based approaches evaluated for analysing the orange sail stripes. After evaluating multiple techniques for identifying and measuring the stripes from high-resolution images, it was demonstrated that this approach provided superior results in accuracy, computational efficiency and capability of handling high-resolution images.

The following sections present a detailed analysis of the decision-making process, highlighting the key factors that led to selecting semantic segmentation for detecting and analysing sail stripes.

Semantic Segmentation vs Instance Segmentation vs Panoptic Segmentation

Semantic segmentation was selected as the optimal approach for this project due to its ability to balance accuracy, computational efficiency, and handle high-resolution images in real-time or near real-time. This method generates a dense, per-pixel classification map, ensuring that every pixel associated with a specific class — in this case, the orange sail stripes — is accurately labelled. This level of fine-grained detail is essential for extracting

aerodynamic measurements such as camber, draft, and twist. While alternative segmentation techniques, such as panoptic or instance segmentation, can distinguish individual objects separately and provide more granular labelling, these methods often introduce increased computational complexity, higher processing demands, and more intricate data annotation requirements. Given the project's objectives, semantic segmentation provides the most effective solution by ensuring precise feature extraction while maintaining efficiency in real-world sailing applications.

From a practical perspective, the visual characteristics of the sail stripes — continuous, elongated regions with minimal within-class variation — make semantic segmentation the most **direct and efficient approach**. Since all orange stripes on the sail belong to the same class and exhibit similar visual properties, there is no inherent need to classify them as distinct instances (as instance segmentation would do). However, to extract shape and curvature parameters, grouping all stripes under a single class is sufficient for isolating and measuring them. Additionally, if further differentiation is needed, a simple post-processing step can be applied to separate or label individual stripes accordingly.

The **pixel-wise nature** of semantic segmentation provides a significant advantage when working with high-resolution images, where fine details play a crucial role in accurate measurement. In the context of sail analysis, even minor variations in line curvature or subtle changes in line thickness can have a measurable impact on calculations related to aerodynamic measurements. In contrast, panoptic and instance segmentation frameworks typically involve additional computational steps, such as instance mask refinement or explicit bounding box regression, which can be particularly demanding for large images. Semantic segmentation, on the other hand, efficiently handles large input dimensions by leveraging multi-scale feature extraction within encoder-decoder network architectures.

Another key advantage is the **relative simplicity of training data requirements**. While all AI-based techniques require labelled examples, semantic segmentation annotations involve labelling regions of interest (in this case, the sail stripes) without needing to label every stripe as a separate instance or track multiple objects of different classes. This can significantly reduce annotation time and ensure faster iteration during model development, which is particularly useful if the project aims to capture varying sailing conditions and needs to incorporate new training data quickly. At the same time, semantic segmentation models can be made more robust through data augmentation, ensuring that they can handle changes in lighting, reflections off the sail, or partial occlusions caused by rigging or sail folds.

Another strength of semantic segmentation is its **comparatively lower annotation complexity**. While all AI-based techniques require labelled training data, semantic segmentation annotations involve marking regions of interest — in this case, the sail stripes — without the need to label each stripe as a separate instance or track multiple objects of different classes. This streamlined annotation process significantly reduces labelling time and accelerates model development, making it particularly beneficial for projects that aim to capture varying sailing conditions and continuously integrate new training data.

Semantic Segmentation vs. Feature Extraction

Feature extraction techniques rely on identifying distinct pixel-level patterns, such as corners, edges, or blobs, often combined with descriptors that facilitate feature matching or classification. While these methods are relatively straightforward to implement, their effectiveness can be significantly compromised when the target object — such as the sail stripe — lacks strong local features or is affected by lighting variations, deformations, or occlusions. Under such conditions, AI-based feature extraction approaches may fail to detect and analyse the sail stripes consistently.

In contrast, semantic segmentation directly classifies every pixel belonging to the orange stripe class, ensuring more robust and reliable detection, even in challenging visual environments. Additionally, because semantic segmentation models are trained end-to-end, they eliminate the need for manually designing or fine-tuning feature extractors. This results in greater adaptability across different sail types, lighting conditions, and stripe variations, making semantic segmentation a more scalable and resilient solution for sail stripe analysis.

Semantic Segmentation vs. Line Detection

AI-based line detection methods effectively identify well-defined linear structures but face challenges when applied to elongated, irregular shapes such as sail stripes. Variations in curvature, thickness, and partial occlusions can lead to fragmented or incomplete detections, requiring additional processing to reconstruct continuous stripe patterns.

However, semantic segmentation offers a more comprehensive and precise approach by classifying every pixel associated with the sail stripes. This method ensures a complete stripe representation in a single step, eliminating the need for post-processing to merge fragmented segments.

Semantic Segmentation vs. Object Detection & Recognition

AI-driven object detection and recognition frameworks often rely on bounding boxes or region proposals to identify objects within an image. While effective for well-defined, discrete entities, these methods can be restrictive when applied to narrow or elongated structures such as sail stripes. Bounding boxes may encompass significant portions of the background or fail to accurately represent the fine-grained geometry of the stripes, complicating subsequent aerodynamic analysis. Object detection approaches are also primarily designed to identify distinct objects rather than continuous patterns, limiting their applicability in this context.

In contrast, semantic segmentation provides a complete mask of the stripe region, preserving its unique geometry and allowing a straightforward extraction of the aerodynamic parameters with greater accuracy, such as camber, twist and draft. Furthermore, bounding box-based methods in large, high-resolution images may become cumbersome and imprecise. In contrast, semantic segmentation provides a more efficient and reliable solution for capturing the whole structure of elongated features like sail stripes.

3.3 Existing AI-Based Approaches for Semantic Segmentation.

This section reviews key AI-based approaches for Semantic Segmentation, such as the SegFormer, DeepLab, Segment Anything Model (SAM) and Fast Semantic Segmentation Convolutional Neural Network (Fast-SCNN) models. Comparing these approaches is essential to gain insights into their capabilities and limitations, providing a foundation for selecting the most suitable method for the thesis objectives.

3.3.1 SegFormer

SegFormer is an advanced Transformer-based architecture designed for semantic segmentation, offering an optimal balance between accuracy, efficiency, and computational scalability [58]. Unlike conventional CNN-based approaches, which rely on convolutional layers for feature extraction, SegFormer utilises a Hierarchical Transformer Encoder (Mix Transformer, MiT) to generate multi-scale feature representations while mitigating the computational overhead typically associated with traditional self-attention mechanisms. Additionally, the model features a lightweight decoder based on Multi-Layer Perceptrons (MLPs), which efficiently aggregates multi-level features without the need for complex post-processing modules.

A key innovation of SegFormer is its ability to eliminate positional encodings, a technique commonly used in Vision Transformers (ViT) that can negatively impact performance when applied to images of varying resolutions. Instead, SegFormer employs a Mix-Feed Forward Network (Mix-FFN), which implicitly integrates depth-wise convolutions to capture positional information. This design enhances the model's robustness to different input resolutions, making it particularly well-suited for real-world applications such as autonomous driving, medical imaging, and robotics.

Unlike previous Transformer-based segmentation models, which directly apply a ViT backbone, SegFormer introduces an overlapping patch-based embedding mechanism to preserve spatial continuity and efficiently process high-resolution images. Furthermore, its hierarchical encoder facilitates global and local feature extraction, enhancing segmentation accuracy across a wide range of datasets.

Hierarchical Transformer Encoder

The MiT encoder in SegFormer functions as a multi-scale feature extractor, addressing a fundamental limitation of traditional Transformers, which often lack hierarchical representation. Unlike ViT, which processes fixed-size non-overlapping patches, MiT is designed explicitly for dense prediction tasks, incorporating several architectural enhancements to improve segmentation performance:

1. **Overlapping Patch Embedding**
    - Standard ViTs divide an image into non-overlapping patches, which can result in a loss of spatial information. However, SegFormer employs overlapping patches, ensuring greater spatial continuity and effective feature representation.
    - This technique preserves local features, making it well-suited for segmentation tasks requiring fine-grained detail.
2. **Hierarchical Multi-Scale Feature Extraction**
    - The MiT encoder progressively reduces spatial resolution while increasing the number of feature channels.
    - Feature maps are generated at four scales (1/4, 1/8, 1/16, and 1/32 of the original image resolution).
    - This design allows the model to retain fine-grained details at higher resolutions while capturing global context at lower resolutions.
3. **Efficient Self-Attention Mechanism**
    - Traditional self-attention mechanisms exhibit quadratic complexity, making them computationally expensive for high-resolution images. In contrast, SegFormer reduces the sequence length using a sequence reduction process, significantly lowering computational cost while preserving contextual information.
    - This approach enables real-time segmentation performance while maintaining high accuracy.
4. **Positional-Encoding-Free Design**
    - Unlike ViT, which uses fixed positional encodings, SegFormer eliminates the need for explicit positional embeddings. Instead, it incorporates depth-wise convolutions in its Mix-FFN, implicitly encoding spatial relationships.
    - This makes SegFormer resolution-agnostic, allowing it to generalise effectively across varying input sizes, enhancing its adaptability to diverse real-world applications.

Lightweight MLP-Based Decoder

Unlike CNN-based segmentation models, which typically utilise complex decoders with multiple convolutional layers, SegFormer employs a lightweight MLP decoder designed to aggregate multi-level features while minimising computational overhead efficiently. The decoder operates through the following key steps:

1. **Channel Unification via MLP Layers:** The encoder generates multiple-level feature maps with varying channel dimensions. The decoder first standardises the channel dimensions using MLP layers to facilitate effective feature fusion, ensuring a consistent representation across different feature scales.
2. **Feature Up-sampling and Fusion:** Feature maps are up-sampled to a common resolution and then connected to integrate fine-grained and coarse-level

information. This fusion process allows the model to preserve local details while capturing broader spatial context, improving segmentation accuracy.

3. **Final Prediction via MLP Layers:** A final MLP layer processes the fused features to generate the segmentation mask. Unlike traditional CNN-based models, this approach eliminates computationally heavy operations, ensuring efficiency without sacrificing segmentation accuracy.

Figure 10 demonstrates SegFormer's Architecture, as explained above.

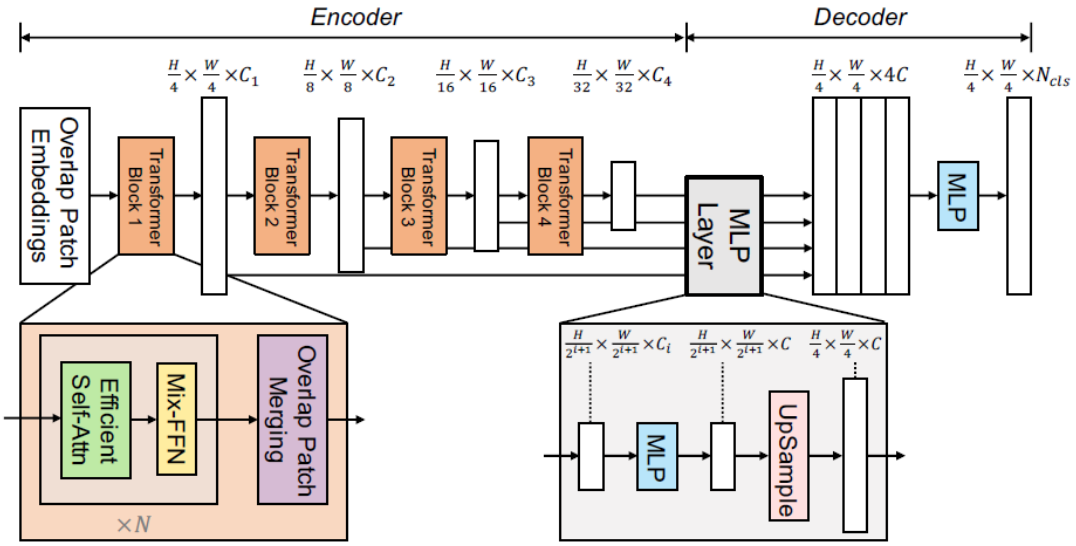


Figure 10 - SegFormer Architecture [58].

SegFormer represents a significant advancement in Transformer-based semantic segmentation by integrating a MiT with a lightweight MLP decoder. This architecture overcomes the inefficiencies associated with traditional self-attention mechanisms, delivering state-of-the-art performance in segmentation tasks. The model's ability to handle varying input resolutions, maintain computational efficiency, and achieve high segmentation accuracy makes it a promising solution for modern segmentation challenges, particularly in resource-constrained environments where real-time processing is essential.

3.3.2 DeepLab

DeepLab is a CNN-based architecture designed for image semantic segmentation, enabling precise object delineation while maintaining computational efficiency [59]. Unlike traditional CNNs, which heavily rely on max-pooling and downsampling, DeepLab introduces several key innovations, including Atrous Convolution, Atrous Spatial Pyramid Pooling (ASPP), and Fully Connected Conditional Random Fields (CRFs). These components enhance the model's ability to capture multi-scale context, preserve fine details, and refine segmentation accuracy.

One of the primary challenges in semantic segmentation is the trade-off between spatial resolution and classification accuracy. Conventional deep networks often experience spatial information loss due to repeated down-sampling operations. DeepLab overcomes this limitation through Atrous (dilated) convolution, which expands the receptive field without introducing additional computational cost. Additionally, ASPP improves the model's ability to process objects at multiple scales, while fully connected CRFs refine boundaries by incorporating global contextual dependencies.

DeepLab has evolved through multiple iterations (DeepLabv1, v2, v3, and v3+), integrating progressively improved techniques to enhance segmentation performance. Compared to Fully Convolutional Networks (FCN) and other CNN-based segmentation models, DeepLab demonstrates superior accuracy, efficiency, and generalisation across several benchmark datasets, including PASCAL VOC 2012, Cityscapes, and ADE20K.

1. Atrous Convolution for Dense Feature Extraction

A fundamental limitation of conventional CNN-based segmentation models is the loss of feature map resolution caused by down-sampling operations such as max-pooling and striding. This reduction in resolution negatively impacts fine-grained segmentation accuracy, as critical spatial details are lost in the process. To address this challenge, DeepLab introduces Atrous Convolution (also known as dilated convolution), which inserts gaps (zeros) between filter weights, allowing it to increase the receptive field without increasing computational complexity or the number of parameters.

- **Controlling the Effective Field of View:** Traditional convolution layers use a fixed receptive field, which can limit their ability to capture large-scale contextual information. Atrous convolution allows DeepLab to enlarge the receptive field dynamically, improving the model's ability to analyse broader spatial relationships while maintaining high-resolution feature representation.
- **Computational Efficiency:** Unlike deconvolution (transposed convolution), which relies on learnable parameters for up-sampling, Atrous convolution maintains efficiency by expanding the receptive field without increasing the model's parameter count. This design significantly improves memory efficiency compared to conventional encoder-decoder architectures.
- **Resolution Preservation:** Standard CNN architectures reduce feature map resolution by a factor of 32 through down-sampling due to repeated down-sampling operations. However, using Atrous convolution, DeepLab minimises this reduction to a factor of 8 while retaining detailed spatial structures. The output segmentation map is then up-sampled using bilinear interpolation to match the original input image size, ensuring high segmentation accuracy.

2. Atrous Spatial Pyramid Pooling for Multi-Scale Context Aggregation

Objects in real-world images appear at various scales, making accurate segmentation challenging. A single fixed receptive field is often insufficient for capturing small-scale details and larger contextual structures. To address this limitation, DeepLab introduces ASPP, a technique designed to enhance multi-scale feature extraction and improve segmentation performance.

- **Multi-Scale Feature Extraction:** ASPP applies parallel Atrous convolutions with different dilation rates, capturing fine and coarse contextual details. Each branch of ASPP processes the feature maps at different resolutions, ensuring robust segmentation across multiple object scales.
- **Parallel Dilated Convolutions:** ASPP efficiently extracts multi-scale information through a pyramid of Atrous convolutions instead of rescaling the input image multiple times. Smaller dilation rates focus on local fine-grained features, while larger dilation rates capture global contextual information, allowing the model to recognise objects at various scales.
- **Reduced Computational Cost:** Traditional multi-scale segmentation methods require processing multiple rescaled copies of an image, leading to significant computational overhead. ASPP provides multi-scale analysis in a single forward pass, greatly enhancing efficiency and processing speed.
- **Improved Generalisation:** ASPP helps the model generalise better in varying lighting, perspective, and environmental conditions, by incorporating multiple receptive fields simultaneously.

3. Fully Connected CRFs for Boundary Refinement

One of the key challenges in deep learning-based segmentation is the accurate delineation of object boundaries. Standard CNN-based segmentation models often produce over-smoothed segmentation masks, leading to poor boundary precision. To address this limitation, DeepLab incorporates Fully Connected CRFs, which refine the segmentation results using probabilistic graphical models.

- **Preserving Object Boundaries:** CNN-based models tend to exhibit invariance to small spatial transformations, which can result in blurry or imprecise edges. CRFs use pairwise pixel relationships to refine segmentation masks, ensuring sharp, well-defined object boundaries.
- **Long-Range Dependencies:** Traditional CRFs model local relationships, limiting their ability to capture global contextual information. But DeepLab employs a fully connected CRF, which accounts for long-range dependencies across the entire image, improving segmentation accuracy for thin and complicated structures such as fences, tree branches, and human silhouettes.

- **Contrast-Sensitive Potentials:** The CRF model incorporates colour, position, and texture features to distinguish between objects with similar visual characteristics. This is particularly useful in crowded scenes or occluded regions.
- **Fast Inference with Mean Field Approximation:** Traditional CRF optimisation is computationally expensive. DeepLab addresses this issue by applying a mean-field approximation, allowing efficient CRF-based refinement without significantly increasing processing time.

Figure 11 illustrates DeepLab's overall architecture.

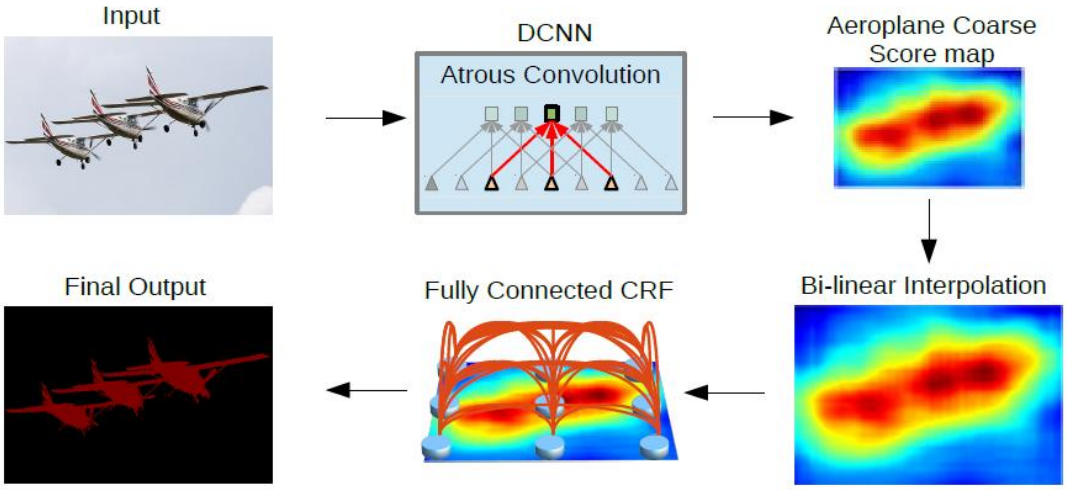


Figure 11 - DeepLab Architecture [59].

DeepLab represents a significant advancement in CNN-based semantic segmentation, integrating Atrous Convolution for dense feature extraction, ASPP for multi-scale analysis, and Fully Connected CRFs for boundary refinement. This combination ensures a high segmentation accuracy approach while maintaining feasibility for real-world applications.

3.3.3 Segment Anything Model

Segment Anything Model (SAM) is a foundation model developed by Meta AI for generalised image segmentation [60], [61]. Unlike traditional segmentation models that require task-specific training, SAM is designed as a promptable segmentation model, capable of segmenting objects in an image without explicit fine-tuning on new datasets. The model was trained on over 1 billion segmentation masks across 11 million images, making it one of the most extensive segmentation datasets.

SAM follows the foundation model paradigm, similar to the approach used in large language models (LLMs) in natural language processing (NLP). It achieves zero-shot generalisation by utilising prompt engineering, allowing users to provide various types of prompts — such as points, bounding boxes, free-form text, and existing masks — to guide segmentation. This

flexibility enables SAM to perform a wide range of segmentation tasks, including instance segmentation and panoptic segmentation.

A core advantage of SAM is its scalability and efficiency. The model is structured around three main components:

- **Powerful image encoder:** Extracts high-level image embeddings, based on a pre-trained ViT.
- **Flexible prompt encoder:** Allows user interaction through diverse input modalities, including points, bounding boxes, masks, and textual descriptions.
- **Lightweight mask decoder**: Efficiently generates segmentation masks in response to the provided prompts.

SAM introduces a novel approach to segmentation by integrating a Transformer-based architecture with a promptable segmentation framework. The core architecture consists of three main components:

1. ViT as the Image Encoder

SAM utilises a high-capacity ViT as its backbone, which is pre-trained using a Masked Autoencoder (MAE) strategy. This hierarchical feature extractor processes images in a manner that allows the model to:

- Generate rich, high-dimensional feature representations of an image, allowing detailed segmentation.
- Precompute image embeddings once, reducing the need for repeated processing.
- Stores embeddings in memory, enabling segmentation to be performed in real-time.
- Facilitates global contextual understanding across the entire image, unlike traditional CNN-based segmentation models like DeepLab or U-Net, which require dense pixel-wise convolutions.

2. Prompt Encoder for Flexible Interactions

A defining feature of SAM is its prompt encoder, which enables interactive and task-agnostic segmentation. This module processes various types of user prompts, including:

- **Sparse prompts:** Points, bounding boxes, or textual descriptions (processed via CLIP-style embeddings).
- **Dense prompts:** Existing segmentation masks or entire regions of interest.

Each prompt is processed individually or in combination, allowing SAM to refine segmentation outputs dynamically. This interactive mechanism facilitates user corrections, improving segmentation results iteratively.

3. Mask Decoder for Efficient Segmentation

The mask decoder in SAM is a lightweight transformer decoder designed to generate segmentation masks from images and prompt embeddings efficiently. This decoder uses:

- **Bidirectional cross-attention** between the image and prompt embeddings to enhance segmentation accuracy.
- **Multi-output segmentation predictions**, when ambiguity in object boundaries arises.
- **Confidence-based ranking** of generated masks using Intersection over Union (IoU) estimation.
- **Class-agnostic segmentation**, eliminating the need for predefined object categories, unlike traditional models.

Figure 12 demonstrates the overall architecture of SAM.

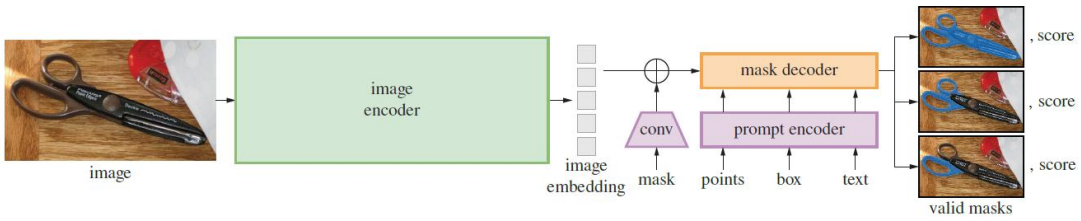


Figure 12 - SAM Architecture [60], [61].

SAM represents a paradigm shift in image segmentation, transitioning from task-specific supervised learning to a generalised, promptable approach. SAM demonstrates strong generalisation capabilities by integrating a ViT-based image encoder, a flexible prompt encoder, and a lightweight mask decoder, outperforming conventional models in interactive segmentation and real-time applications.

3.3.4 Fast Semantic Segmentation Convolutional Neural Network

Fast-SCNN is a deep learning model designed explicitly for real-time semantic segmentation of high-resolution images [62]. In contrast to traditional segmentation architectures that rely on complex encoder-decoder frameworks or multi-stage processing pipelines, Fast-SCNN optimises the segmentation process to achieve high accuracy with minimal computational cost. This design makes it particularly well-suited for applications such as autonomous driving, robotics, and embedded systems, where processing speed and memory efficiency are paramount.

One of the main innovations in Fast-SCNN is the **"Learning to Down-sample"** module, which enables the simultaneous extraction of spatial details at high resolution and global contextual information at a lower resolution. This dual approach enhances the model's

ability to outperform other real-time segmentation models, such as ICNet, BiSeNet, and ContextNet, in terms of speed and efficiency.

1. Learning to Down-sample Module

The Learning to Down-sample module in Fast-SCNN extracts low-level spatial details while efficiently reducing the resolution of input features. This module comprises:

- A standard convolutional layer (Conv2D) for initial feature extraction, capturing foundational spatial information.
- Two Depthwise Separable Convolution (DSConv) layers reduce the computational cost by separating spatial and channel-wise convolutions.
- Stridden convolutions to progressively reduce the spatial resolution while preserving essential features.

This module ensures the **preservation of critical low-level spatial information** while minimising computational complexity, as an efficient alternative to traditional deep encoder structures.

2. Global Feature Extractor

Once the input features are down-sampled, Fast-SCNN employs a Global Feature Extractor to capture high-level semantic context. This module is inspired by the bottleneck residual blocks of MobileNet-V2, which incorporate:

- Efficient representation of complex object relationships, ensuring robust segmentation performance.
- Depth-wise Separable Convolutions to reduce the number of parameters and overall computational cost.
- Pyramid Pooling Module (PPM) at the end to capture multi-scale context information, enhancing the model's ability to handle objects of varying sizes and scales.

3. Feature Fusion and Classification Module

The final stage of Fast-SCNN is the Feature Fusion and Classification Module, which combines low-level spatial features with high-level contextual features to generate accurate segmentation maps. This module includes:

- Feature Fusion Module (FFM), which performs element-wise addition of different resolution feature maps to combine spatial and contextual information effectively.
- Depth-wise Separable Convolution layers to refine feature representations for improved accuracy.
- A lightweight classification head, utilising pointwise convolutions to generate the final segmentation mask.

Figure 13 illustrates the overall architecture of Fast-SCNN, as explained above.

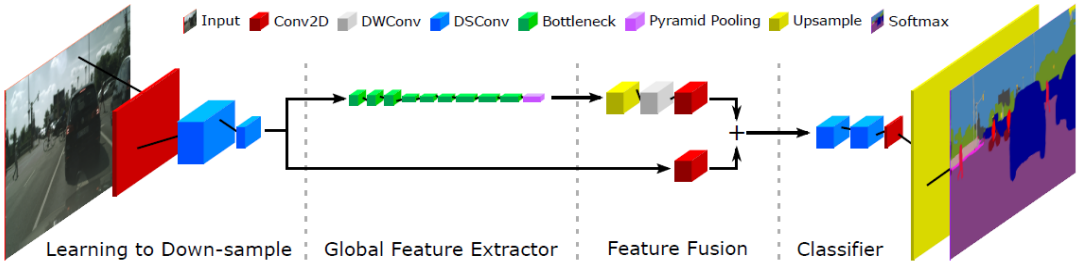


Figure 13 - Fast-SCNN Architecture [62].

With a lightweight architecture of only 1.11 million parameters, Fast-SCNN achieves a mean Intersection over Union (mIoU) of 68.0% on the Cityscapes dataset while operating at an impressive 123.5 frames per second (FPS) [62]. These metrics make Fast-SCNN one of the fastest real-time segmentation models currently available. Fast-SCNN is specifically designed to be memory-efficient and adaptable to low-power hardware. This makes it an ideal solution for resource-constrained environments where real-time performance is critical.

3.3.5 Comparative Analysis and Selection of the Best AI-Based Approach

After evaluating multiple advanced semantic segmentation methods, **SegFormer** emerged as the most suitable solution for this project. SegFormer offers a compelling balance of **accuracy, computational efficiency, and the capability of handling high-resolution images in near real-time.** A key advantage of SegFormer lies in its hierarchical attention-based architecture, which effectively integrates local and global contextual information. This capability ensures the precise detection of fine-grained details, such as the thin orange sail stripes, while maintaining high fidelity. Importantly, this strong performance does not come at the cost of excessive computational demands. SegFormer's lightweight decoder and configurable backbone variants help manage resource consumption, ensuring feasibility for onboard systems that demand efficiency. Additionally, the project benefits from SegFormer's robust open-source community, which provides access to pre-trained models, sample code, and continuous improvements, facilitating further development and refinement.

SegFormer vs DeepLab

Computational costs are a primary consideration when comparing SegFormer and DeepLab for semantic segmentation. While DeepLab achieves high segmentation quality on academic benchmarks, its application to detecting thin lines in high-resolution images often necessitates a heavier backbone and additional multi-scale processing strategies. These requirements can increase computational costs and reduce efficiency, particularly in hardware-constrained environments such as those found aboard a sailboat.

DeepLab typically relies on high-performance Graphics Processing Unit (GPU) resources to process large images effectively and prevent memory bottlenecks. In settings where computational power and energy availability are limited, DeepLab may need to downscale input resolution or adopt smaller backbones, which can compromise fine-grained segmentation accuracy — a critical factor in detecting narrow orange sail stripes. Additionally, while DeepLab is an open-source framework, its integration and optimisation for real-time maritime applications can be complex, requiring significant modifications to align with the constraints of embedded GPU systems.

In contrast, SegFormer offers a more balanced approach by optimising accuracy and speed. Its lighter computational footprint makes it inherently better suited for processing high-resolution images on moderate hardware, without sacrificing segmentation detail. The Transformer-based architecture of SegFormer, along with its flexible model variants, enhances its adaptability to onboard systems and mobile platforms, making it a more practical choice for real-time sail analysis.

Overall, the efficiency, scalability, and lower resource demands of SegFormer make it a more viable solution for embedded maritime applications. In contrast, DeepLab's higher computational requirements and complex optimisation needs can present challenges in such environments.

SegFormer vs SAM

SAM demonstrates high accuracy across diverse objects and scene types, but its general-purpose design and extensive pretraining requirements result in significant computational costs during inference [63]. Processing large images with SAM can be memory-intensive and computationally demanding, which poses challenges for achieving near real-time feedback in a sailboat environment.

While SAM is open-source and offers unique prompting methods that enable the segmentation of arbitrary objects, this feature becomes less relevant in a system designed to autonomously detect narrow orange sail stripes without requiring continuous human input. Additionally, SAM’s reliance on large-scale pretraining and its high parameter count make it less suitable for resource-limited environments, as its deployment in onboard systems would lead to increased power consumption and reduced efficiency.

Although SAM is capable of zero-shot segmentation, its computational overhead can be excessive for specialised tasks such as detecting the orange stripes in the sail. In contrast, SegFormer’s architecture is specifically designed to handle the fine-grained segmentation of narrow structures with lower memory consumption and faster inference times. Its targeted approach and ability to fine-tune task-specific datasets make it a more practical and efficient solution for aerodynamic measurements in maritime applications.

SegFormer vs Fast-SCNN

Fast-SCNN is designed for minimal computational cost and high efficiency. It is an appealing option for real-time processing on resource-constrained edge devices, such as those used on a sailboat. However, in prioritising speed, Fast-SCNN often compromises accuracy, particularly when handling large images that capture thin, high-contrast sail stripes under variable environmental conditions. The model may struggle to precisely resolve subtle edges, especially when subjected to aggressive downsampling to meet strict real-time constraints.

Although Fast-SCNN is an open-source implementation, its emphasis on speed can result in segmentation outputs that may miss or blur narrow sail stripes, thereby affecting the accurate estimation of critical sail parameters such as camber, draft, and twist. In contrast, SegFormer's advanced attention-based architecture effectively preserves fine details while balancing computational efficiency and segmentation accuracy. While SegFormer imposes slightly higher computational demands than Fast-SCNN, it remains sufficiently lightweight to operate on moderate hardware, ensuring prompt inference and reliable detection of thin sail lines in high-resolution imagery.

Ultimately, SegFormer's capability to process large-scale inputs with fine detail retention, without significantly compromising speed or necessitating specialised equipment, makes it a more practical and effective solution for maritime deployment in sail analysis applications.

Comparative Table analysis

Table 1 provides a comparative overview of these algorithms, highlighting their strengths and potential challenges in the context of the thesis's objectives.

Table 1- Comparative Analysis and Selection of the Best AI-Based Approach.

Criteria	DeepLab	SAM	Fast-SCNN	SegFormer	Why SegFormer is the better choice?
Accuracy	Offers segmentation on benchmarks but may require large backbones to capture small details accurately.	Robust across diverse object categories but requires prompts; not specifically optimised for detecting the orange stripes.	It is inadequate in simpler scenes; it can suffer when objects are very thin or in varying lighting conditions due to its lightweight design.	Uses hierarchical Transformers to capture fine-grained details (e.g., thin lines) and global context.	Attains high accuracy without depending on excessively heavy backbones, making it more flexible for custom tasks like orange line detection on sails.

Capability of Handling Large Images	It can process large images but often requires significant GPU memory; it might use lower input resolution to cope with high-resolution images.	It can handle large inputs but often incurs very high GPU memory usage and longer inference times; it is not ideal for real-time, high-resolution use.	Handles large images by down-sampling aggressively for speed, risking the loss of fine detail like narrow lines.	Designed to efficiently manage global and local features with attention mechanisms, retaining detail even at high resolutions.	Delivers good segmentation quality on high-resolution images without excessively straining hardware resources, outperforming others in capturing thin features in large-scale settings.
Computational Costs	Atrous convolutions and large backbones can demand powerful GPUs/TPUs, and can be computationally heavy for real-time edge deployment.	Large model size and extensive pretraining lead to considerable memory and processing requirements; specialised hardware is often needed.	Very light on resources, but achieves that by cutting back on fine detail.	Allows the selection of smaller or larger variants; matches model size to available GPU power and memory constraints.	Maintaining a strong balance between computational demands and segmentation quality makes it simpler to run on a modest GPU without drastic performance loss.
Real-Time performance	It can achieve near real-time speeds if heavily optimised or run on high-end GPUs, but maintaining detail is challenging under these constraints.	Not primarily designed for near real-time inference; optimising for speed can degrade its ability to segment “anything” with minimal prompts.	Known for real-time operation on resource-limited devices, though thin objects may be missed due to limited representational power.	Relatively fast inference for a transformer model; the lightweight MLP decoder and hierarchical attention make it viable for near real-time operation.	SegFormer remains fast enough for onboard analyses while preserving the resolution and detail needed to track orange tape lines, striking a pragmatic balance that the others struggle to achieve in this specific scenario.

The comparative analysis of the four AI-based segmentation algorithms — SegFormer, DeepLab, SAM, and Fast-SCNN — reveals diverse strengths and weaknesses that serve

different application requirements. This comparative analysis underscores the importance of aligning algorithm selection with the thesis's objectives to achieve optimal performance.

3.4 Conventional-Based Image Processing Approach

This thesis uses a traditional image processing technique as a reference for comparison to more evidently reveal the strengths and weaknesses of the proposed technique of AI-based semantic segmentation. This approach, previously implemented using Open-Source Computer Vision Library (OpenCV), is used to reveal the performance of the traditional techniques when applied to segmenting orange lines on a sailboat sail and extracting aerodynamic parameters.

The algorithm was not developed as part of this thesis, but is included for comparison purposes.

3.4.1 System Architecture and Pipeline

The traditional detection method is fully integrated into the complete sail analysis system architecture that comprises three primary elements: a tablet, a camera, and a software workflow that executes on the tablet.

Figure 14 illustrates a GoPro camera mounted on a sailboat capturing high-definition images of the sail and sending them to the processing unit (a tablet) through an HTTP-based connection. The tablet has the necessary software elements — computer vision code in OpenCV and GoPro camera communication in Open GoPro — to parse the arriving images and collect aerodynamic parameters. The client-server system architecture that works in the background keeps the system autonomous of external computer resources and robust in offshore deployments by doing the heavy lifting of the analysis locally in the tablet.

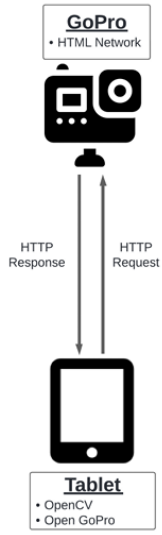


Figure 14 - Traditional-Based Approach System Architecture [2].

Once the images reach the tablet, they pass through a well-structured software pipeline delineated in Figure 15. The pipeline consists of three stages:

1. **Take Photo:** The system requests and receives an image from the GoPro.
2. **Detection Algorithm:** The algorithm locates the sail lines.
3. **Calculations:** The detected lines compute aerodynamic parameters such as camber, draft, and twist.

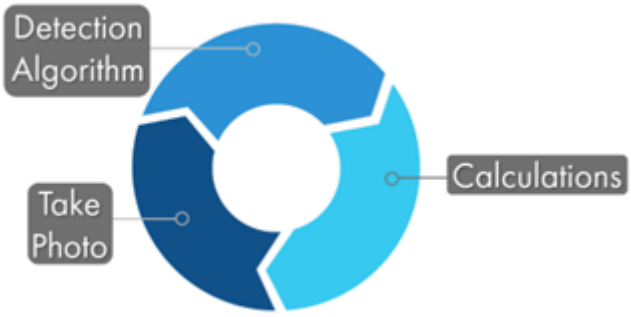


Figure 15 - Traditional-Based System Pipeline [2].

With this approach, a controlled and direct comparison is possible for this thesis: the classical algorithm and the AI model are put into the same pipeline with the same images and evaluated with the same parameter extraction methods.

Thus, the role of the traditional image processing approach in architecture is to provide a comparative reference that outlines the advantages and disadvantages of the deterministic approach against AI-based semantic segmentation.

3.4.2 Algorithm

The traditional algorithm's internal structure follows a step-by-step sequence of classical image processing transformations, each stage transforming the image in preparation for the next. Such structured flow is illustrated in Figure 16, where the flow from raw image input to the end output line point is graphically demonstrated.

The stages can be described as follows:

- **Image Cut:** The first step reduces the perception area to the sail region while cutting off unnecessary regions such as the horizon, water, or mast; reducing the noise and accelerating the process.
- **Camera Calibration:** As in GoPro cameras, wide-angle lenses create distortions that bend straight lines. Calibration compensates for these distortions because of known intrinsic parameters, so lines are geometrically consistent in the image.
- **Hue Adjustment:** Since the sail lines are painted in orange, the algorithm converts the image to the HSV colour space and enhances the hue range corresponding to orange tones. This helps bring lines from the sailcloth and background into better visibility.
- **Blur:** A Gaussian blur is applied to smooth noise with high frequencies (such as small shadowing or fabric texture). Minimising false positives when thresholding helps preserve the larger sail line structures.
- **Thresholding:** The thresholding is done for the filtered image by selecting pixels in the given range of orange. Pixels in the same colour as the line are marked as foreground, while the others are suppressed.
- **Dilation:** Since lines may appear broken as light varies, the morphological dilation operation thickens and unites fragmented line regions to provide smoother shapes.
- **Contours:** The algorithm extracts contours from the binary mask. Each contour is a potential line or a section of a line.
- **Noise Filter:** To remove irrelevant detections, contours are filtered based on their size, shape, and elongation. Small blobs or irregular regions are discarded, leaving only contours consistent with line geometry. To suppress unwanted detection, edges are filtered based on size, shape, and elongation. Irregular regions or small gaps are discarded while contours consistent with line geometry are preserved.
- **Point Reduction:** The remaining contours are simplified into ordered sets of points along every sail line. This reduces computational complexity while preserving line geometry.
- **Output Points:** The algorithm's end product is the collection of line points that may be sent to the parameter extraction phase. Chords may be specified across the sail, polynomials may be determined to fit line curves, and aerodynamic parameters may be calculated from here.

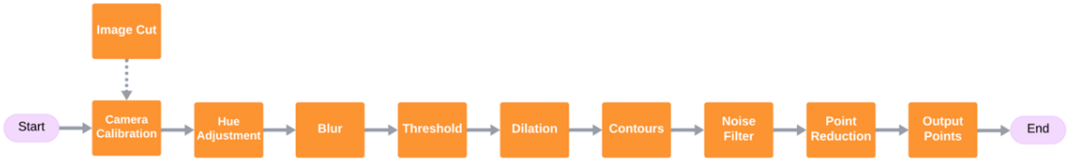


Figure 16 - Traditional-Based Algorithm [2].

3.4.3 Strengths and Weaknesses

The traditional approach is fast, lightweight, and can efficiently run on embedded systems such as tablets. Its deterministic rules make it transparent and tunable, which is suitable for understanding how the pipeline performs under different image conditions.

However, it is also influenced by external factors such as light or shadow, or objects that are in a similar colour. Because performance is created on fixed thresholds and handcrafted rules, performance may degrade under varying sailing conditions.

3.4.4 Comparative Purpose in this Thesis

In this thesis, the traditional algorithm is used as a comparison baseline. Comparing the result with that obtained from the AI-based semantic segmentation technique, it is possible to:

- Quantify the improvements in segmentation accuracy from AI.
- Compare processing times of both algorithms and evaluate real-time feasibility.
- Assess robustness in changing lighting and environmental conditions.
- Evaluate whether measurable gains justify the additional computational complexity of AI.

The comparison is provided in Chapter 6, where both approaches use the same dataset and compare their performance through consistent metrics.

To conclude this chapter, Figure 17 presents the type of output the traditional image processing technique offers. The identified lines are noted on the sail surface, while their respective aerodynamic parameters — camber, draft, and twist — are provided numerically in diverse sail heights (top, middle, and bottom).

Camber Top	Draft Top	Twist Top
17,83%	42,11%	9,26°
Camber Mid	**Draft Mid**	**Twist Mid**
17,32%	42,23%	7,35°
Camber Bot	**Draft Bot**	**Twist Bot**
13,58%	43,12%	4,73°

Figure 17 - Results of the Traditional-Based Approach.

4 System Architecture

Developing a robust system for estimating aerodynamic parameters of a sail relies on a properly structured architecture that integrates several elements ranging from data acquisition to model training, inference and parameter computation. This chapter describes the architecture, components, and data flow between them. Figure 18 represents the overall system architecture, although this thesis only focused on the parts shown in orange.

The architecture has seven main components:

- **Image Acquisition:** Raw sail images and videos are collected using onboard cameras, such as a GoPro, and transmitted to an edge device (e.g., a tablet).
- **Model Training:** Previously acquired images are pre-processed and used to train the segmentation model. After training, the best-performing model is deployed into the inference pipeline within the local deployment environment.
- **Model Inference:** The trained model processes newly acquired data to create segmentation masks and overlays for the sail.
- **Parameter Calculation:** Based on the model inference outputs, aerodynamic parameters such as camber, draft, and twist are calculated.
- **User Interface (UI):** Visual overlays of detected lines and chord points are displayed for sailors to interpret directly.
- **Text Interface:** Aerodynamic parameters are presented in a structured numerical format, supporting human interpretation, automation, and possible integration with navigation systems. The UI and the Text Interface run in parallel to provide complementary outputs.
- **Local Storage:** Results from the UI and Text Interface are stored in CSV format, ensuring traceability, persistence, and offline evaluation.

The process begins with **Image Acquisition**, which collects raw sail data. From there, the workflow splits into two paths: **Model Training**, which develops the segmentation model, and the **Local Deployment Environment**, which includes **Model Inference** and **Parameter Calculation**, where the trained model is applied to new data. The results are delivered simultaneously through the **UI** and the **Text Interface**, providing visual overlays and numerical outputs. Finally, all results are saved in **Local Storage** for persistence and offline analysis.

This thesis focuses on the main modules highlighted in orange: **Model Training, Model Inference, Parameter Calculation, UI, Text Interface, and Local Storage.** Image Acquisition is treated as supporting infrastructure, developed in a previous thesis [2], and is therefore not included in the technical contributions of this work.

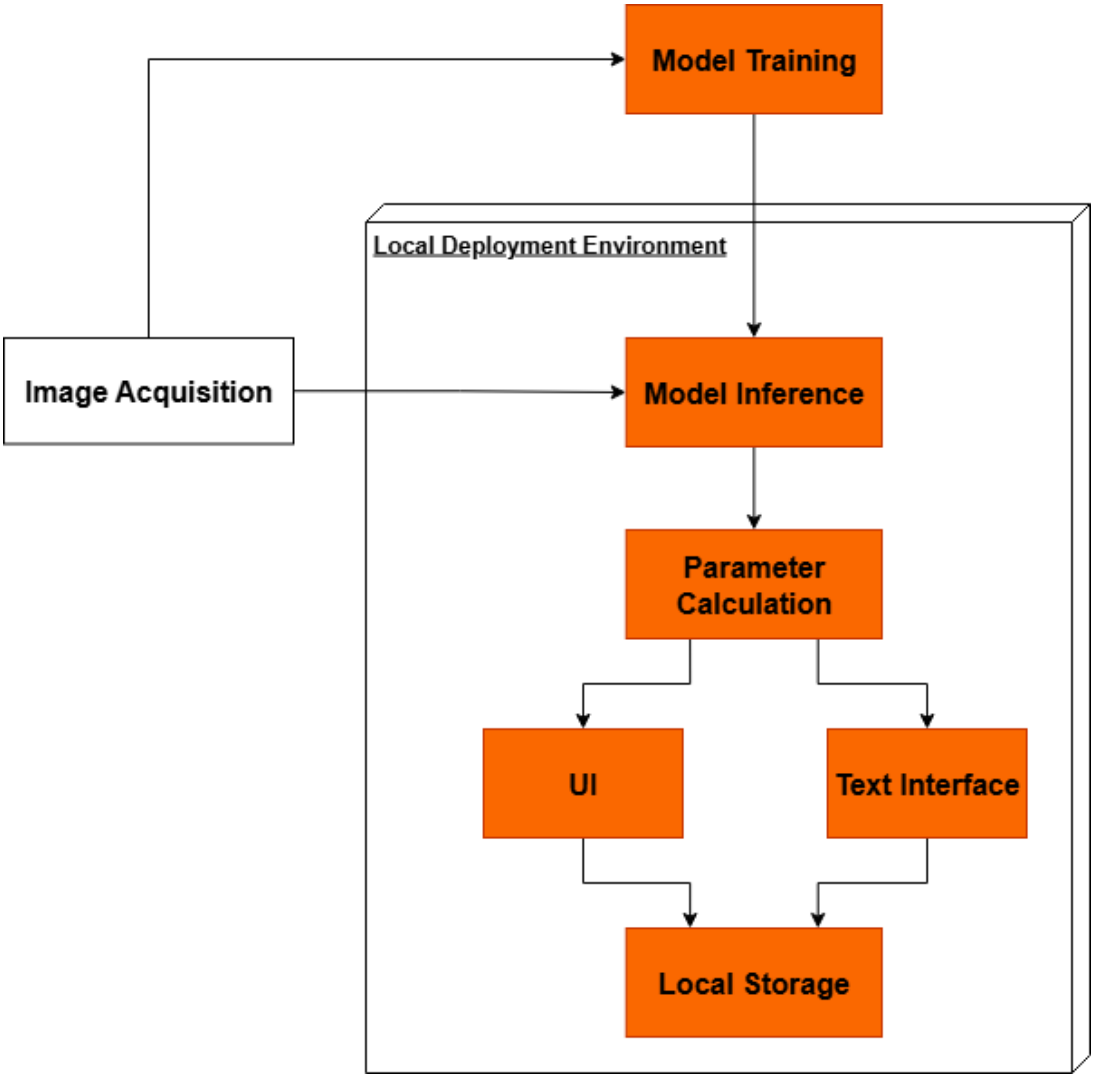


Figure 18 - System Architecture Design.

Although Model Inference and Parameter Calculation are different processes, their outputs are delivered to the user at the same time. This is why the UI and the Text Interface (Figure 18) run in parallel.

The UI presents an overlay inference that shows the detected lines, chord points, and aerodynamic parameters on the original sail image. This gives sailors an immediate and clear visual understanding of performance characteristics. Meanwhile, the Text Interface generates a structured numerical output of the three aerodynamic parameters for each frame. This text format allows results to be logged, sent, or combined with external navigation systems, making it easy for humans to read and machines to use.

Finally, all outputs are saved systematically in the Local Storage module, where results are kept in CSV format. This lightweight and efficient storage system ensures data is preserved and suits the workflow for limited onboard environments.

Chapter 5 gives more details about the high-level architecture.

4.1 Model Training

The Model Training block (Figure 18) is a key system part. It affects how accurately and reliably the system can detect sail lines during inference. This workflow is broken into six stages, which explain how to prepare the segmentation model for deployment. Figure 19 provides a breakdown of the processes contained within the Model Training block.

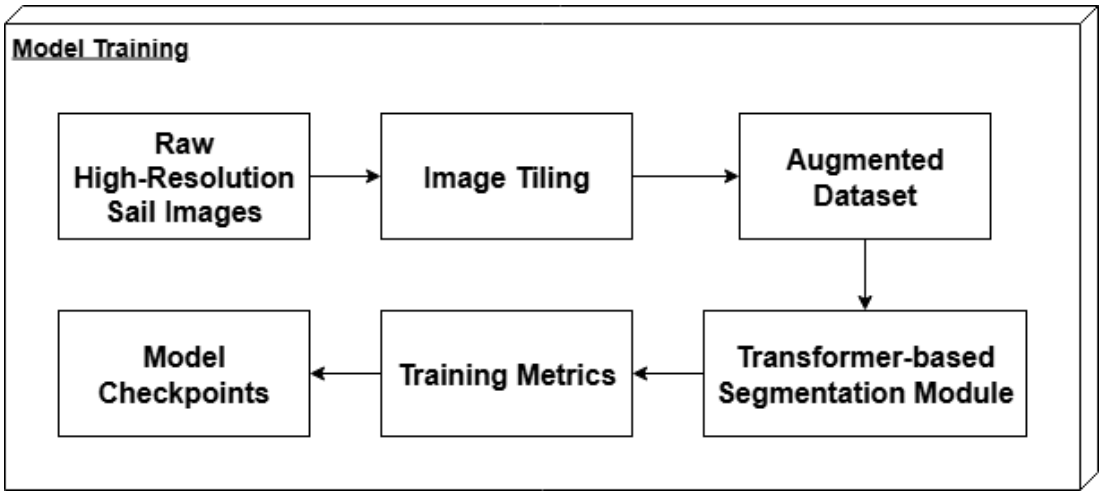


Figure 19 – Model Training Design Architecture.

The process starts with collecting raw, **high-resolution sail images**. This level of detail is essential for capturing the thin orange lines on the sail.

However, it also creates a significant computational challenge. Most deep learning frameworks cannot process such large images directly because of memory and runtime limits. To solve this problem, the images are passed through an **image tiling step**, split into overlapping tiles of 2000×2000 pixels. This tiling method makes training manageable while keeping sail lines continuous across the tile edges, makes it computationally feasible, and augments the dataset by effectively increasing the number of available training samples. This choice compromises the necessity of preserving fine detail in the lines and the realities of GPU processing. Tiling has the following advantages:

- **Preservation of detail:** Thin lines are well preserved and visible without being blurred by aggressive downscaling.
- **Increased sample count:** Each high-resolution image produces multiple training samples, effectively augmenting the dataset.
- **Efficiency:** Smaller tiles require less memory and allow for larger batches or more extensive augmentations.

The practical details of the tiling process will be explained later in Chapter 5.1.1, which will cover implementation aspects.

Once tiling is complete, the dataset is expanded through **augmentation**. This stage is essential because sail images vary significantly due to different lighting, sail angles, sea surface reflections, and other environmental factors. **Augmentation** adds controlled changes like rotations, flips, and brightness shifts. This helps improve the model's ability to handle real-world conditions. The augmented dataset is then used to train a **Transformer-based segmentation model (SegFormer)**. Unlike typical CNN architectures, Transformers can capture long-range connections within the image. This feature is handy for detecting continuous lines like sail lines, which can span large areas of the frame. Chapter 5.1.2 discusses the implementation of this module in more detail.

The training process uses **metrics** chosen to deal with the strong imbalance in the dataset. About 90% of all pixels belong to the background class, while around 10% correspond to sail lines. This imbalance makes conventional accuracy misleading. During testing, some models that reported accuracy above 95% failed to detect lines, simply predicting background everywhere. Several complementary metrics were used during training and validation to address this issue.

- **Global accuracy and Loss:** This is a general indicator that measures the proportion of correctly classified pixels across the entire image, but is heavily influenced by background performance.
- **Line-specific accuracy:** To reflect line quality more directly, this metric focuses only on line pixels to show the model's ability to identify the thin orange lines of the sail.
- **IoU:** measures the degree of overlap between predicted lines and the proper annotations, penalising false positives and false negatives.
- **Dice coefficient:** serves a similar role but emphasises smaller structures, making it particularly suitable for thin sail lines.
- **F1-score:** balances precision and recall. Precision reflects the model's ability to avoid false detections of lines, while recall captures its ability to identify all true line pixels.

In addition to these evaluation metrics, the loss function itself guides training. The loss is created as a weighted combination of Dice Loss and Focal Loss. This design addresses class imbalance by focusing on sail line detection while decreasing the effect of easily classified background pixels. Chapter 5 details the implementation of this process in the training procedure.

IoU, Dice, and F1 provide a more reliable evaluation of the model's performance under the severe imbalance imposed by the dataset. The training logs in Chapter 6.3 report results for global accuracy, line-specific accuracy, IoU, and the Dice coefficient, as these offer the most relevant insights into the model's effectiveness. Although the F1-score is not directly plotted in the logs, it conceptually supports these measures by clarifying the trade-off between detecting more line pixels (recall) and avoiding false detections (precision).

Finally, the process is supported by **model checkpoints**, where the model's parameters are saved at regular intervals. This prevents loss of progress during interruptions and helps

select the best-performing model for deployment. When this process ends, the system provides a trained segmentation model ready for use in the **Model Inference** stage.

4.2 Model Inference

The Model Inference module (also represented in Figure 18) is one of the operational stages of the system. At this point, the trained segmentation model is used to derive higher-level aerodynamic parameters from new data. The processes inside the Model Inference block are illustrated in Figure 20.

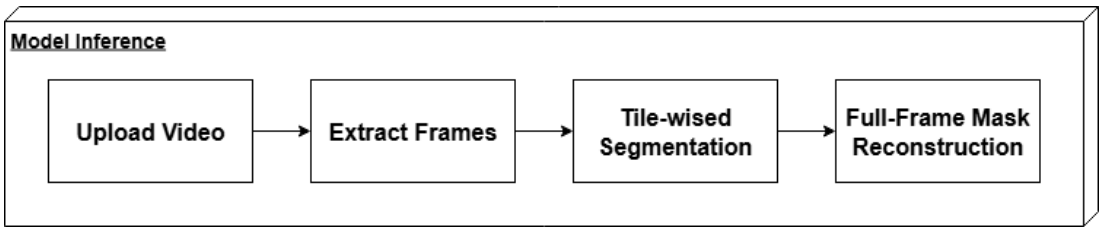


Figure 20 - Model Inference Design Architecture.

The inference process starts with **uploading a video** that shows the sail in real operating conditions. Videos offer richer temporal context than static images, making them especially useful for analysing the dynamic behaviour of the sail.

From the video, **frames are extracted** at set intervals, balancing between temporal resolution and computational efficiency. Each frame is divided into tiles using the same method as in the training phase. This keeps large images manageable while ensuring that sail lines remain continuous across boundaries. This implementation is presented in Chapter 5.2.2.

The trained segmentation model processes each tile separately, creating partial masks of the detected sail lines, which are then reconstructed into complete full-frame segmentation masks. Further explanation of this implementation can be found in Chapters 5.2.3 and 5.2.4.

Once the Model Inference stage ends, the workflow moves directly into the Parameter Calculation stage (Chapter 4.3). At this point, the segmented outputs from inference are used as inputs for further processing. This step-by-step handover ensures that the raw detection results become useful performance metrics for the sail.

4.3 Parameter Calculation

Once the Model Inference stage ends, the Parameter Calculation module, shown below, derives the aerodynamic characteristics of the sail from the segmented output produced in the Model Inference stage.

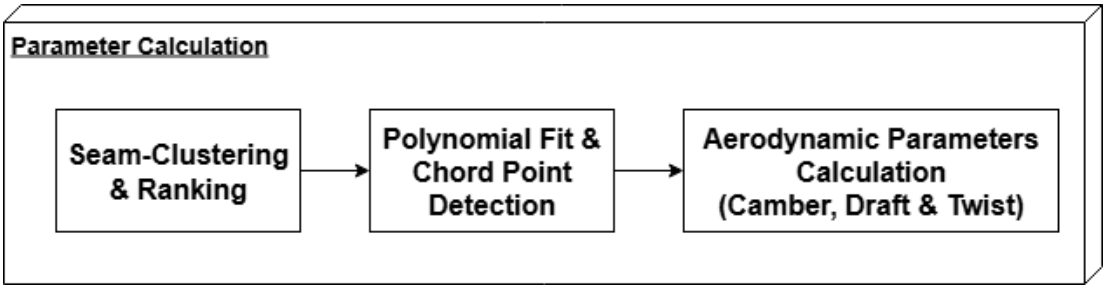


Figure 21 - Parameter Calculation Design Architecture.

Once the full-frame segmentation mask is ready, the next step is refining the detected line segments. A **clustering algorithm** groups redundant detections and reconnects fragmented line sections, producing coherent and continuous line representations. A **ranking strategy** filters out noise and false detections to enhance reliability by prioritising the most relevant lines.

The clustered lines are then fitted with **polynomial functions** to approximate smooth curves that accurately reflect the sail's geometry. At this stage, **chord points** — key reference points traversing from the sail's leading edge to its trailing edge — are identified. These chord points act as geometric anchors for calculating aerodynamic parameters.

Based on these geometric references, the system **calculates the three main aerodynamic parameters: Camber, Draft, and Twist.**

Chapter 5.2.5 explains the algorithms and computational methods that support these calculations and shows how they fit into the workflow.

5 System Implementation

This chapter presents the proposed system's implementation, centring around a SegFormer-based semantic segmentation model designed to detect three narrow orange lines on a sail. These sail lines serve as crucial structural markers, and their geometry enables the calculation of key aerodynamic parameters such as camber, draft, and twist. The system workflow involves preparing and augmenting the dataset, followed by model training to ensure reliable performance under different lighting conditions and sail deformations. Once the model is trained, it is used during inference to analyse video frames through a tiled segmentation approach. Post-processing steps for sail line refinement, chord line extraction, and polynomial curve fitting follow this. The entire system has been deployed on a GPU-equipped laptop, featuring integrated pipelines for data handling, training, and inference — delivering a comprehensive end-to-end solution for accurate sail shape analysis using AI-based algorithms.

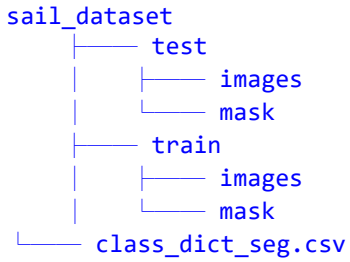
5.1 Model-Training

This thesis aims to train a SegFormer-based semantic segmentation model to detect three narrow orange lines of a sail. The choice of this model is justified by its efficiency and strong performance in segmentation tasks, as discussed in Chapter 3.3.5. These sail lines are functional because they are critical structural elements of a sailboat, whose actual geometry allows us to extract aerodynamic parameters such as camber, draft, and twist. These lines are likely to manifest under varied lighting, sail shapes, and angles of the cameras in realistic situations. Therefore, the challenge is to train a robust, accurate, and generalizable model for unseen situations. The code developed for the same has a complete end-to-end pipeline.

5.1.1 Dataset Preparation

Dataset Structure and Organisation

The dataset was organised to allow clean separation between training and testing data. This helps in ensuring performance reports are unbiased and accurately reflect the capability of the model to generalise in unexpected circumstances. The chosen directory structure has different images and masks subdirectories in the test and train sets, and a CSV file defines the class names against numerical values.



This human-readable and machine-friendly structure allows easy incorporation with PyTorch's data loading mechanisms.

One key step in preparing the dataset was making the class definitions consistent. Only two classes are specified for this task: the background and the line. Such a small class set makes the segmentation task easy and the class imbalance more prominent because the background covers most pixels.

The CSV mapping was executed with the following code:

Algorithm 1: Build Class Mappings from CSV

Input: a CSV file (class_dict_seg.csv) inside ROOT_DIR
Output: two mappings; (id2label → maps numeric class IDs to class names) & (label2id → maps class names to numeric class IDs).

1 Read the CSV file class_dict_seg.csv from ROOT_DIR into a data table df.
2 Extract the column "name" from df and store it as classes (classes = df["name"]).
3 **Convert** classes into a dictionary (id2label = classes.to_dict()):
    // Keys = row indices (0, 1, 2, …)
    // Values = class names.
    // Store this as id2label.
4 **Reverse** the mapping to create label2id (label2id = {v:k for k,v in id2label.items()}):
    // Keys = class names
    // Values = row indices.
5 **Return** id2label and label2id.

Listing 1 - Build Class Mappings from CSV.

The listing code above shows how to create two dictionaries that link numeric class IDs to class names. First, the program reads the CSV file and extracts the name column, which gives a list of all class labels. Each label corresponds to its row index, serving as the numeric ID. From this, the first dictionary, id2label, maps IDs to names. Then, the mapping gets reversed to create label2id, which links names back to IDs. Returning both dictionaries allows the project to easily switch between numeric IDs, used by the model, and human-readable labels, used for interpretation.

Binary Mask Generation

The segmentation masks mark sail line locations and are processed to achieve consistent labelling. The raw masks often contain varying grayscale values due to export formats or

annotation tools. These are regulated to a binary format represented in the CSV file in the dataset:

- **0** for background.
- **1** for line.

This mask was generated using the following code execution:

Algorithm 2: Load and Normalise Mask

Input: a grayscale mask image file.
Output: a cleaned binary mask with pixel values of 0 (background) or 1 (line).

1 Read the mask image from the file in grayscale mode (m).
2 **IF** a pixel value equals 255, set it to 1.
3 **IF** a pixel value is not 0 or 1, set it to 0.
4 **Return** the normalised mask m.

Listing 2 - Load and Normalise Mask.

The algorithm above shows how to load and clean a mask image for processing. First, the mask is read in grayscale mode. Any pixel with a value of 255 is normalised to 1. Any pixel value that is not 0 or 1 is set to 0. This ensures the final mask has only two values: 0 for the background and 1 for the line.

Then, a morphological dilation is applied to the line regions with a 3×3 kernel. As reinforced earlier, this small thickening of the lines helps balance small annotation errors, giving the model a more continuous and learnable signal during training time. This implementation is demonstrated in Listing 3.

Algorithm 3: Mask Dilation

Input: a binary mask m and a kernel self.dilate_kernel.
Output: a modified binary mask where the line regions (value 1) are enlarged, making them thicker and more connected.

1 Take the binary mask m.
2 Apply morphological dilation using the kernel self.dilate_kernel.
    // Each 1 pixel in the mask expands outward to its neighbourhood defined by the kernel.
    // The parameter iterations=1 means dilation is applied once.
3 Store the result back into m.
4 **Return** the dilated mask.

Listing 3 - Mask Dilation.

The algorithm above improves a binary mask by enlarging the regions marked as lines. It does this by using morphological dilation with a predefined kernel. Each pixel with a value of 1 expands into its surrounding area. After one iteration, the result is a mask where the line regions look thicker and more connected. This improves their continuity before further processing.

The binary masks of the dataset were produced automatically from the original sail video footage using a C++ code adapted from a previous model developed in the traditional-based approach. This tool was included in the dataset preparation pipeline to prevent manual annotation and improve efficiency, productivity, and reproducibility.

The program extracts a fixed number of evenly spaced frames from the input video. These frames are then subjected to image processing steps executed using the OpenCV library. The frame is first converted from the RGB into the HSV colour space to facilitate direct manipulation of the hue component. The hue component is then adjusted to seek a strong chromatic contrast between the lines and the surrounding sail fabric, making them more detectable.

Next, the high-frequency noise is suppressed using a Gaussian blur, and the thresholding operation is applied to separate candidate line regions from the background. Another colour-based range filter sets the selection by removing pixels with improper hue and saturation values outside the ranges of the lines. Morphological dilation is applied to connect fragmented line segments and improve contour continuity.

Some mask contours are extracted from the processed mask and ranked according to their spatial coverage. To minimise the inclusion of irrelevant information, only the largest line-like contours are retained with an upper limit of three per frame. Such contours are drafted into a final binary mask where white and black pixels denote the line and background.

These derived masks become the ground-truth labels of the training and test subsets. Reusing and fine-tuning a previously proven algorithm and tuning its parameters to the sail footage's specific light and colour conditions made it possible to create high-quality and stable annotations ready for supervised model training.

Image Resolution and Tiling

These images were captured at a high resolution and preserved excellent detail. However, it is impractical to train on images of that kind (as mentioned in Chapter 4.1, Model Training block).

To overcome this, the images and masks were divided into 2000×2000 pixels tiles using a code adapted from a GitHub repository [64] and ensures complete coverage of the image while keeping tile sizes manageable:

Algorithm 4: Image Tiling.

Input: a folder of images (IMAGE_DIR) and crop dimensions (HEIGHT, WIDTH).
Output: a set of cropped image tiles saved to an output folder and the total number of crops (count).

```
Load all images from IMAGE_DIR → images.
Set count = 0.
FOR each img in images:
    Get img_height, img_width
    vertical = ceil(img_height / HEIGHT)
    horizontal = ceil(img_width / WIDTH)
    Set start_v = 0.
    FOR each vertical slice v = 0 TO vertical - 1:
        Set start_h = 0.
        FOR each horizontal slice h = 0 TO horizontal - 1:
            IF start_h + WIDTH > img_width → set start_h = img_width - WIDTH.
            IF start_v + HEIGHT > img_height → set start_v = img_height - HEIGHT.
            Crop region = img[start_v : start_v+HEIGHT, start_h : start_h+WIDTH].
            Save crop, increase count.
            Move right: start_h = start_h + WIDTH.
        Move down: start_v = start_v + HEIGHT.
END FOR (all images processed).
Return the total number of crops (count).
```

Listing 4 - Image Tiling.

This algorithm divides large images into smaller, evenly sized tiles. Each image comes from the input folder, and the number of vertical and horizontal slices depends on the crop dimensions. The algorithm moves through the image row by row and column by column. At each step, it extracts a crop of the specified height and width, adjusting at the edges to stay within the image boundaries. Each cropped tile is stored in the output folder, and a counter tracks the total number of crops produced. After processing all the images, the algorithm provides the total count.

In addition to this external tiling step, the training pipeline further sampled 512×512 crops from the tiles, implemented through Line-Biased Cropping.

5.1.2 Data Augmentation Strategy

An overall augmentation pipeline was employed to improve the robustness and ability of the model to generalise using geometric, photometric, and colour-based transformations. The augmentations explained below were chosen to counter the sail line segmentation's particular challenges of class shortage, changing light and wind conditions, and variation in the positioning of the cameras or sail orientation.

Geometric augmentations included horizontal and vertical flips simulating tack and sail orientation changes, allowing the model to handle mirrored situations. Affine transformations included controlled changes of scale, translations, and rotations, preparing

the model to handle changes of angles of the cameras and the natural deformations of the sail. Random cropping was also used to focus the model on local image regions; however, due to the low spatial density of the lines, a specific cropping procedure was specified to avoid the loss of target structures too often. These are demonstrated below:

This approach, called **Line-Biased Cropping**, constituted one of the strongest contributions in the augmentation procedure. In regular random cropping, most samples hold no line pixels, offering very limited or no adequate training signal. Line-Biased Cropping mitigates this through probabilistically centring crops around line pixels so that a considerable percentage of the cropped fragments include the corresponding features. This avoids lost computation, boosts the convergence rate, and improves the model's performance.

Line-Biased Cropping was implemented as follows:

Algorithm 5: Line-Biased Cropping

Input: a binary segmentation mask (mask) together with the crop dimensions (height, width) and a probability (p).
Output: the top-left coordinates {y0, x0} of a valid crop window of size height × width.

```
Read mask and get image size: H, W = mask.shape.
Generate a random number r ∈ [0,1).
IF r < p AND mask has at least one positive pixel:
    Find positions of all positive pixels: (ys, xs) = where(mask == 1).
    Pick a random index idx.
    Set crop centre: (cy, cx) = (ys[idx], xs[idx]).
    // Compute tentative top-left:
        y0 = cy − height // 2
        x0 = cx − width // 2
    // Clamp values so the crop stays inside:
        y0 = clamp(y0, 0, H − height)
        x0 = clamp(x0, 0, W − width)
ELSE (no biased pixel chosen):
    // Randomly sample valid coordinates:
        y0 = randint(0, H − height)
        x0 = randint(0, W − width)
Return {y0, x0}.
```

Listing 5 - Line-Biased Cropping.

Listing 5 selects a crop window from an image, focusing on areas with line pixels. It starts by generating a random number after checking the mask size. With a probability p, and only if the mask has positive pixels, the crop is centred around one of these pixels. The chosen pixel becomes the centre of the crop, and the top-left corner is calculated to ensure the crop remains within the image boundaries. If no positive pixel is selected, either because the probability test fails or the mask is empty, the crop window is placed at random valid coordinates. The final output is the top-left position of the crop.

Photometric improvements simulated variability in the environmental scene and sensor characteristics. Colour jittering changed brightness, contrast, saturation, and hue to mimic the impact of different environmental illumination and weather conditions. Additional hue, saturation, and value shifts added variability observed during training and resistance to variation across camera models and conditions. A Gaussian blur was added to introduce small amounts of defocus, habituating the model to images captured under motion or imperfect focus.

All of the augmentations mentioned above are implemented in the following code:

Algorithm 6: Define Data Augmentation Transforms

Input: crop size (CROP_SIZE).
Output: two transformation pipelines: one for training (train_tf) and one for validation (val_tf).

1 Set the crop size: CROP_SIZE = 512x512.
2 Define training transformations (train_tf):
  Line-biased crop of the image and mask.
  Random horizontal flip.
  Random vertical flip.
  Random affine transformation (scaling, translation, rotation).
  Random colour jitter (brightness, contrast, saturation, hue).
  Random hue, saturation, and value shift.
  Random Gaussian blur.
3 Define validation transformations (val_tf):
  Resize image and mask to CROP_SIZE × CROP_SIZE.
4 **Return** train_tf and val_tf.

Listing 6 - Define Data Augmentation Transforms.

The algorithm above outlines the steps to prepare training and validation data. Random training changes, such as biased cropping, flips, affine transformations, colour adjustments, and blurring, are applied for training. These variations help the model perform better by introducing different versions of the same data. The process is straightforward for validation: images and masks are resized to a standard crop size, ensuring consistency during evaluation. It ends up with two transformation pipelines, one for training and one for validation.

Last but not least, a **synthetic recolouring** procedure was added to decrease the model's reliance on colour information. A set of sail line pixels in chosen training images was artificially recoloured to a highly saturated orange. Forcing the model to distinguish lines from shape and texture information rather than employing their inherent colour increases its flexibility under a wide range of natural conditions where the line colours could differ. The following code presents the synthetic recolouring procedure:

Algorithm 7: Synthetic Recolouring Augmentation

Input: an image (img), a binary mask (mask), and a fraction of pixels to recolour (fraction).
Output: a modified image where a subset of the white pixels inside the mask are recoloured to orange.

```
Convert the input image img from RGB to HSV colour space → hsv.
Find all pixel coordinates (ys, xs) where mask == 1.
IF there are any such pixels:
    Calculate how many to recolour: n = fraction × total number of masked pixels.
    Randomly select n pixel indices from the masked region.
    Change the hue channel of the selected pixels to represent orange 20.
    Set the saturation channel of the selected pixels to the maximum value of 255.
Convert the modified HSV image back to the RGB colour space.
Return the recoloured image.
```

Listing 7 - Synthetic Recolouring Augmentation.

Listing 7 creates synthetic data by changing the colour of part of the mask region in an image. First, the image is converted into the HSV colour space, which helps adjust colour properties more easily. All pixels in the mask are identified, and a portion of them is randomly selected for changes. These chosen pixels are turned orange by adjusting their hue and increasing their saturation. Finally, the image is converted back to RGB format and returned, now featuring artificially created orange areas to enhance training diversity.

5.1.3 Loss Functions and Class Imbalance

The severe imbalance between background and sail line pixels, approximately 90% to 10%, makes choosing a loss function crucial. This issue was introduced in Chapter 4.1, Model Training block, which discusses it in detail along with evaluation metrics.

To overcome this, the training loss is a weighted combination of two specialised losses:

- **Dice Loss:** helps balance the contribution of both classes, no matter how many pixels they have.
- **Focal Loss:** focuses on the line pixels that are harder to classify.

The implementation details of these custom-loss functions can be found in the listing codes below:

Algorithm 8: Dice Loss

Input: predicted logits (logits) and the ground truth labels (targets).
Output: a scalar Dice loss value, where lower values indicate better overlap between predictions and targets.

1 Apply softmax to the logits and extract the probability for the positive class → probs.
2 Create a validity mask → mask = (targets != ignore_index).
3 Convert ground truth to binary tensor → t = 1 where target == 1, else 0.
4 Apply the validity mask to predictions → p = probs × mask.
5 Compute the intersection → inter = sum(p × t).
6 Compute the denominator → denom = sum(p) + sum(t).
7 Compute Dice score → dice = (2 × inter + self.smooth) / (denom + self.smooth).
8 Compute Dice loss → 1 - dice.
9 **Return** the Dice loss.

Listing 8 - Dice Loss.

The algorithm above measures how well the predicted segmentation matches the ground truth. First, the predicted logits go through a softmax function to get class probabilities, focusing on the positive class. The ground truth labels are changed into a binary form, and any ignored areas are masked out. Using this information, the intersection is computed between predictions and targets, as well as their combined size. The Dice score is twice the intersection divided by the sum of the two areas, including a small smoothing factor to prevent division by zero. Finally, the Dice loss is defined as one minus the Dice score, meaning lower values show better agreement between prediction and ground truth.

Algorithm 9: Focal Loss

Input: the predicted logits (logits) and the ground truth labels (targets).
Output: a scalar Focal loss value, which down-weights easy examples and focuses learning on harder ones.

1 Rearrange logits so each pixel has a class probability vector → reshape into 2D form.
2 Flatten targets into a 1D vector.
3 Compute per-pixel cross-entropy loss → ce.
  Use optional class weights.
  Ignore pixels marked with ignore_index.
4 Convert cross-entropy loss to predicted probability of the true class → pt = exp(−ce).
// Apply Focal weighting:
5 loss = ((1 − pt) ^ gamma) × ce.
  // This reduces the weight of well-classified (easy) samples.
6 Take the mean across all pixels.
7 **Return** the Focal loss.

Listing 9 - Focal Loss.

As reinforced earlier, this algorithm aims to tackle class imbalance by concentrating training on more challenging examples. First, the predicted logits are reshaped so that each pixel matches a probability vector, and the target labels are flattened. Next, the standard per-

pixel cross-entropy loss is calculated, allowing for class weighting and the option to ignore certain pixels. From this, the probability given to the true class is extracted. A focal weighting factor is then applied. This reduces the impact of well-classified, easy pixels while emphasising the difficult ones. Finally, the weighted losses are averaged across all pixels, resulting in the final Focal loss value.

The combination utilises Dice Loss at the regional line level and Focal Loss at the fine-grained level for the minority class accuracy.

5.1.4 Training Procedure

The training procedure follows the architecture and design principles outlined in Chapter 4. Each epoch has a training phase where the model learns from augmented data and a validation phase where it assesses its performance on unseen data.

The AdamW optimiser over trainable parameters (decoder) with weight decay to regularise the head and limit overfitting is deployed, and mixed-precision training accelerates computation and reduces memory usage, enabling larger crop and batch sizes. A learning rate scheduler monitors validation loss and reduces the learning rate when improvement stabilises, allowing for finer weight adjustments as training progresses.

Mixed precision training and loss combination were applied during each step using this listing code:

Algorithm 10: Forward and Backwards Pass with Mixed Precision

Input: a batch of images (x) and labels (y).
Output: updated model with gradients applied using Dice + Focal loss under mixed-precision training.

Enable mixed precision training context (autocast).
// Forward pass through the model:
    Compute logits from input batch x.
    Upsample logits (interpolate) to match the spatial size of the labels y.
Compute total loss: Weighted sum of dice_loss and focal_loss.
Scale the loss using GradScaler → prevents underflow in mixed precision.
Backpropagate scaled loss (backward).
Update model parameters with the optimiser (step).
Update the scaler for the next iteration (update).
Output: model weights updated for this training batch.

Listing 10 - Forward and Backwards Pass with Mixed Precision.

Freezing the Encoder

The MiT backbone encoder is frozen in this training setup, meaning its weights are not updated during training. This decision was made because:

- The encoder has been pretrained on large, diverse datasets (e.g., ImageNet-like datasets), providing strong general-purpose features.
- The dataset for this project, while high-quality, is relatively small compared to datasets used in general-purpose training.

Training only the decoder reduces the number of parameters that must be updated, making it fast and reducing the risk of overfitting. The encoder freezing was achieved with:

Algorithm 11: Freeze Encoder Parameters

Input: model's encoder parameters.
Output: an encoder with disabled gradients, meaning its weights will not be updated during training.

1 Loop through all parameters p in the model's encoder.
2 **FOR** each parameter, **SET** requires_grad = False.
  This tells PyTorch not to compute gradients for these parameters.
3 **Output:** the encoder weights are frozen (they remain fixed during training).

Listing 11 - Freeze Encoder Parameters.

The algorithm above ensures that a model's encoder part stays the same during training. It goes through all encoder parameters and turns off their gradient computation.

This strategy effectively exploits the pretrained encoder as a very competent feature extractor while adapting the decoder for the specific task of line segmentation.

At the end of each epoch, checkpoints are saved to keep track of progress and allow for recovery in case of interruption of the training procedure. This checkpointing matches the Model Checkpoint block in Chapter 4.1 Model Training, ensuring the best-performing model for use in inference is retained.

5.2 Model Inference and Parameter Extraction

This chapter presents the working phase in which the trained segmentation model is executed to analyse sail images and extract aerodynamic parameters. The process begins with loading the fine-tuned SegFormer model, setting it into a ready-for-inference state, and extracting the sailing video frames from a stored sailing video. A tiled segmentation technique examines these frames for sail line identification with high accuracy, independent of the image resolution. Post-processing techniques are then employed to smooth out the continuity of the detected lines, followed by the identification of correct chord lines and polynomial fitting of curves to describe the curvature of the lines. This geometric representation calculates three fundamental parameters — camber, draft, and twist — for multiple sail sections.

5.2.1 Model Loading and Initialisation

The inference pipeline begins by bringing the trained SegFormer model into action. The model that has been selected, mit-b1, is a lightweight and high-performing encoder–decoder network that is specially optimised for semantic segmentation tasks. The model is initialised through the correct number of output labels (line and background), specified in the dataset preparation step. Hence, the segmentation correctly distinguishes the pixels between the sail line and the background. A class name and a mapping of numerical identifiers are also initialised for consistency with the training step.

The model weights are loaded from a checkpoint containing all parameters learned during training. A "SegformerImageProcessor" library performs required preprocessing operations such as resizing, normalisation, and converting into PyTorch tensors. Then, the model is in evaluation mode, i.e., it will not compute gradients and will provide stable, deterministic output. Inference runs on the most powerful available device — a GPU, if available — for fast execution of the segmentation and handling of high-resolution video frames within a certain time (depending on the duration of the video). The implementation of this procedure is shown in Listing 12.

Algorithm 12: Load SegFormer Model

Input: the target computing device (device) and the class mapping file (class_dict_seg.csv).
Output: a trained segmentation model (model) and a feature extractor (fe) ready for inference or further training.

Read the CSV file class_dict_seg.csv from the dataset directory.
Extract the class names and build two mappings:
    id2label: numeric ID → class name.
    label2id: class name → numeric ID.
Initialise the SegFormer image processor (fe).
Load a pretrained SegFormer model with:
    Architecture specified by ARCH.
    Number of labels = number of classes.
    Label mappings (id2label and label2id).
    Options to handle size mismatches and reshape the encoder output.
    Move the model to the given device (CPU or GPU).
IF a checkpoint file exists:
    Load the model's saved weights from the checkpoint.
Set the model to evaluation mode (eval).
Return both the model and the feature extractor.

Listing 12 - Load SegFormer Model.

The algorithm above explains how to prepare a SegFormer model for segmentation tasks. It starts by reading the class mapping file and creating dictionaries that connect numeric IDs to their class names. These mappings go into the SegFormer setup to ensure predictions match the dataset labels. An image processor is initialised to handle input preprocessing, and a pretrained SegFormer model is loaded with the right architecture and number of

output classes. The model is moved to the specified device, CPU or GPU for computation. If a checkpoint is available, previously trained weights are restored, and the model is set to evaluation mode. Finally, the model and the feature extractor are returned and ready for inference or further training.

5.2.2 Frame Extraction from Camera Video

The input is a video recording of the sail in use. Instead of processing every frame, which would be computationally inefficient, a set of fixed frames — 20 frames — is selected at spaced intervals (duration of the video divided by 20). This uniform spacing ensures that selected frames capture the whole range of conditions in the video, from sail shape variations to lighting variations to points of sail.

The frames are extracted by OpenCV's "VideoCapture" API, providing very accurate positioning within the video stream. The selection process works by calculating the total frame count, dividing it into equal pieces, and selecting one representative frame from each piece. The frames are kept at their natural resolution, so as not to lose any geometric features needed for proper line detection and parameter measurement afterwards, as seen in Figure 22.

Figure 22 - Example of an Extracted Frame from a Video.

This implementation is executed using the following code:

Algorithm 13: Extract Frames from Video

Input: a video file path (path) and the number of frames to extract (n).
Output: a list of extracted frames (frames).

```
Open the video at the path using cv2.VideoCapture.
IF the video cannot be opened → raise an error.
Get the total number of frames in the video.
Compute n evenly spaced frame indices across the video.
Initialise an empty list of frames.
FOR each index in the selected frame indices:
    Move the video reader to the given frame.
    Read the frame.
    IF the frame is valid, append it → frames.
Release the video capture object.
Return the list of extracted frames.
```

Listing 13 - Extract Frames from Video.

Listing 13 explains how to get a fixed number of evenly spaced frames from a video file. The process starts by opening the video with OpenCV's capture tool and checking if it can be accessed. The total number of frames in the video is retrieved. From this, a set of evenly distributed frame positions is calculated. The reader moves to the correct position for each selected index and extracts the frame. Valid frames are saved in a list, and any invalid attempts are skipped. After processing all frames, the video capture is released, and the list of extracted frames is returned for further use.

5.2.3 Segmentation Processing

A tiled inference approach is taken because the input frames are potentially much larger than the average segmentation models can handle in one pass. Each frame is decomposed into fixed-size tiles (512×512 pixels per tile in this project), allowing the system to work at high resolutions without breaking memory limits. This step also improves the segmentation quality by maintaining detail in smaller regions.

Each of the tiles is passed through the preprocessing pipeline of "SegformerImageProcessor", preparing it exactly how the SegFormer model expects. The model outputs raw logits per class that are later upscaled by bilinear interpolation to achieve the original shape of the tile. The pixel-wise class that is most probable is selected to build a discrete segmentation map.

Once all tiles for a frame have been processed, they are reassembled into a final segmentation mask. This mask holds the sail lines and background regions predicted by the model, and the line pixels are uniquely labelled for further isolation in later steps. The core functionality is provided in Listing 14.

Algorithm 14: Segment Frames and Compute Line Properties

Input: a list of video frames (frames), a segmentation model (model), a feature extractor (fe), and the computing device (device).
Output: an overlay image (original frame + segmentation mask visualisation) and computed line properties (props).

```
Initialise an empty list (out).
FOR each frame in the list of frames:
    Convert the frame from BGR to RGB.
    Get frame height H and width W.
    Initialise a prediction mask pred with zeros (H × W).
    // Divide the frame into tiles:
    Loop over the frame in steps of TILE_SIZE.
    Extract each tile (tile) using coordinates (y1:y2, x1:x2).
    Preprocess the tile with the feature extractor (fe) and send to device.
    // Without gradients:
        Run the tile through the segmentation model to get logits.
        Upsample logits back to the tile size (bilinear interpolation).
        Fill in the corresponding region of pred with predicted labels.
Compute geometric properties of the segmented line using
compute_line_properties(pred).
    // Create an overlay visualisation:
    Copy the original frame.
    Build a colour mask (green) for predicted line pixels.
    Blend the original frame and mask with transparency.
Append (overlay, props) to the result list out.
Return out.
```

Listing 14 - Segment Frames and Compute Line Properties.

The algorithm above uses the segmentation model on video frames to extract important geometric features of the detected lines. Each frame is first converted to the correct colour format and set up with a blank prediction mask. Because frames can be large, they are divided into smaller tiles, which the feature extractor and segmentation model process individually. The model outputs are resized to fit the tile dimensions and then combined into the complete prediction mask. Once the whole mask is created, the geometric line properties are calculated. For visualisation, a copy of the original frame is blended with a coloured mask highlighting the predicted line areas. Both the overlay image and the computed properties are saved, and the entire set of results is returned after processing all frames.

5.2.4 Sail Line Segmentation and Post-Processing

The mask also undergoes some post-processing operations to smooth out the structures of the lines. The line class is initially binarised and creates a binary mask for which the pixels of the lines are white and other pixels are black (1 for white/line and 0 for black/background). Morphological closing is also applied to close small gaps that may appear due to imperfections during the segmentation process.

This refined mask identifies lines by invoking OpenCV's “findContours” method, returning a set of disjoint line fragments. These fragments take part within the same real line but may be severed by noise or occlusions in the scene. To recover continuous lines, a union–find merging algorithm is called to gather fragments whose ends are within a given pixel distance. This merge step is needed to recover a complete line profile.

The merged clusters are filtered based on their point density, and only meaningful line shapes are potentially suitable for analysis. In instances with more than three clusters of lines, those closest to the horizontal centre of the frame are chosen. This would be the sail's top, middle, and bottom regions (ordered from top to bottom of the image), the most important for aerodynamics analysis. The listing code X for these steps is available in the Annexe.

Each detected sail line has a chord line defined by the cluster's leftmost and rightmost. They represent the edges of the lead, and the line trails from the frame's perspective. This chord serves as a geometric baseline for later calculations.

A **third-degree polynomial** is fitted through the line's pixel coordinates to approximate the sail line's curvature. This approximates the curvature by smoothing out tiny irregularities while maintaining the general geometric trend. The **derivative** of the polynomial is used to find the location at which the curvature of the line has a maximum — the **maximum camber point**. From this point on, a perpendicular line to the chord is calculated to compute the camber height and, from that point on, the aerodynamic parameters. The code is demonstrated in Listing 15 on the next page.

Algorithm 15: Compute Line Properties

Input: A binary mask image (mask) together with the parameters: the number of lines to extract (num_lines), the degree of the polynomial fit (poly_deg), the maximum distance threshold for merging fragments (merge_thresh), and the minimum number of points required to consider a cluster valid (min_cluster_pts).
Output: A list containing exactly num_lines, where each sail line includes its chord start point (first), chord end point (last), maximum camber point (max_pt), perpendicular intersection point (intercept_pt), and the aerodynamic metrics camber (percentage), draft (percentage), and twist (degrees).

Extract the height and width of the mask.
Binarise the mask so that line pixels become white (255) and all others black (0).
Apply a morphological close operation with a small kernel to connect nearby pixels and close micro-gaps.
// Extract contours from the mask and build fragments:
    Ignore contours with a very small area.
    FOR valid contours, reshape into a list of points and compute their convex hull.
    Find the leftmost (xmin) and rightmost (xmax) positions, and compute the average y-coordinates at those positions (y_min, y_max).
    Define chord endpoints first = (xmin, y_min) and last = (xmax, y_max).
    Ensure first.x < last.x by swapping if necessary.
    Store each fragment with its points and chord endpoints.
    IF no fragments are found, return an empty list.
// Merge nearby fragments using union–find:
    Compare all fragment pairs by the distances between their endpoints.
    IF the minimum distance is smaller than merge_thresh, merge them into the same group.
// Build clusters from the merged fragments:
    FOR each group, combine all points and compute the average x-coordinate (center_x).
    Add to the list of all clusters.
    IF the cluster has at least min_cluster_pts points, add it to the list of large clusters.
// Select candidate clusters:
    IF at least num_lines large clusters exist, use them.
        Use them.
    ELSE
        Use all clusters.
    IF still no candidates exist, return an empty list.
Sort the candidate clusters by how close their center_x is to the horizontal centre of the image, and select the top num_lines.
// Process each selected cluster:
    Determine chord endpoints as the points with minimum and maximum x-coordinates.
    Fit a third polynomial of degree poly_deg through all cluster points and compute its derivative.
    Compute the chord slope m and intercept b.
    Calculate the maximum camber point (max_pt) using tangent matching.
    Find the perpendicular intersection point (intercept_pt).
    Compute camber percentage, draft percentage, and twist angle.
    Store all these values along with the chord endpoints and polynomial coefficients.
Sort the resulting sail lines from top to bottom according to their vertical position (center_y) and return the final list.

Listing 15 - Compute Line Properties.

This algorithm extracts aerodynamic line properties from a binary mask image created before. It starts by converting the mask to highlight line pixels and applies morphological closing to connect small gaps. From the cleaned mask, it detects contours and breaks them into fragments, each defined by its chord endpoints between the leftmost and rightmost points (leech and luff represented in Chapter 2.1). Very small contours are removed, and nearby fragments are combined into larger groups using a distance-based union-find method. Clusters of enough size are then formed, and only the strongest candidates, based on their number of points and location, are retained.

Chord endpoints are found for each selected cluster, and a 3rd degree polynomial is fitted through the points to model the line. This polynomial calculates geometric features, including the maximum camber point, its perpendicular intersection with the chord, and the chord slope. From these, aerodynamic metrics are calculated: camber as a percentage of chord length, draft as a percentage of depth, and twist as an angular measure. Each line is saved with its key points and metrics. Finally, the detected lines are sorted from top to bottom. This ensures that the output matches the exact line positions: top, middle, and bottom. This way, the returned results clearly align with the expected order of lines along the sail.

The outcome of the segmentation is shown in Figure 23.

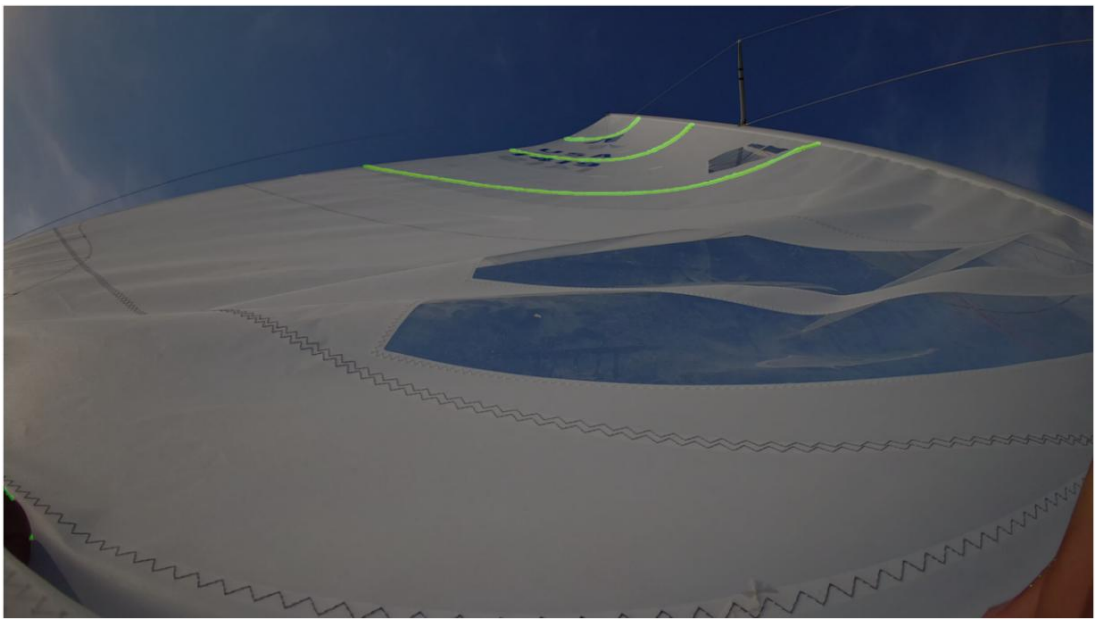


Figure 23 - SegFormer Mask Segmentation.

5.2.5 Parameter Computation and Visualisation

Having established the line geometry, the camber, draft, twist and maximum camber point are calculated for each line, as seen in the Listing 16, Listing 17, Listing 18 and Listing 19, respectively.

The equations for the Camber, Draft, Twist and Maximum Camber Point calculations are demonstrated in Chapter 2.2.1, 2.2.2, 2.2.3 and 2.2.4, respectively.

Algorithm 16: Calculation of Camber

Input: max_pt = (x_m, y_m) — point of maximum camber; intercept_pt = (x_o, y_o) — perpendicular projection of max_pt onto chord; first = (x_1, y_1) and last = (x_2, y_2) — chord endpoints

Output: camber as a percentage of chord length (camber_percent).

1. Compute chord length: chord_len = $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$
2. Compute camber length: cam_len = $\sqrt{((x_m - x_0)^2 + (y_m - y_0)^2)}$
3. Calculate: camber_percent = (cam_len / chord_len) × 100.
4. **Return** camber_percent.

Listing 16 - Calculation of Camber.

Listing 16 measures the curvature of the sail by comparing the distance from the maximum camber point to the chord with the overall chord length. First, it calculates the length of the chord between its two endpoints. Then, it finds the perpendicular distance from the maximum camber point to the chord. The ratio of these two distances, expressed as a percentage, gives the camber. A higher percentage indicates a deeper curvature relative to the chord.

Algorithm 17: Calculation of Draft

Input: first = (x_1, y_1) and last = (x_2, y_2) — chord endpoints; intercept_pt = (x_o, y_o) — perpendicular projection point

Output: the draft position measured from the trailing endpoint (last) as a percentage of chord length (draft_right_percent).

1. Compute chord length: chord_len = $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$
2. Compute distance from last to intercept: dist = $\sqrt{((x_2 - x_0)^2 + (y_2 - y_0)^2)}$.
3. Calculate: draft_percent = (dist / chord_len) × 100.
4. **Return** draft_percent.

Listing 17 - Calculation of Draft.

The algorithm above determines where the maximum camber occurs along the chord. It starts by calculating the total chord length and then measures the distance from the trailing edge (the last chord endpoint) to the perpendicular projection of the camber point. Dividing this distance by the chord length and converting it to a percentage gives the draft. This value shows how far along the chord the maximum sail depth is, which is crucial for aerodynamic performance.

Algorithm 18: Calculation of Twist

Input: the slope of the chord line m.
Output: the twist angle (twist_deg) in degrees.

1 Compute absolute slope: |m|.
2 Convert slope to angle: $\theta = \tan^{-1}(|m|)$
3 Convert to degrees: twist_deg = $\theta \times \left(\frac{180}{\pi}\right)$.
4 **Return** twist_deg.

Listing 18 - Calculation of Twist.

This algorithm quantifies the angular rotation of the chord relative to a horizontal reference. It begins with the slope of the chord and converts it into an angle using the arctangent function. The result is then expressed in degrees, reflecting the twist. This angle indicates how much the sail's orientation tilts, influencing how airflow interacts with it.

Algorithm 19: Calculation of Maximum Camber Point

Input: a polynomial curve poly(x) that represents the sail line, its derivative dpoly(x), and the chord endpoints first = (x_1, y_1) and last = (x_2, y_2). It also uses a constant NUMBER_OF_POINTS to decide how many subdivisions are made when searching for the camber point.
Output: the point of maximum camber max_pt = (x*, y*), which is the point on the curve where the tangent slope is the closest to the slope of the chord.

1 Compute the slope of the chord connecting the two endpoints: $m = (x_2 - x_1) + (y_2 - y_1)$
// Define the search interval along the x-axis:
2 Start of interval: start = x_1.
3 End of interval: end = x_2.
4 Step size: step = (end − start) / NUMBER_OF_POINTS.
// Initialise the variables for the search:
5 closest_x = start → the x-position of the current best match.
6 min_disc = | m − dpoly(start) | → the difference between the chord slope and the tangent slope at the start point.
7 x = start → the current position for evaluation.
// Perform the search by iterating over the interval:
8 **FOR** i = 0 **TO** NUMBER_OF_POINTS:
9 Compute slope mismatch at the current x: disc = | m − dpoly(x) |.
10 **IF** this difference disc is smaller than the current best min_disc:
11 Update min_disc = disc.
12 Update closest_x = x.
13 Move to the next x position: x = x + step.
14 After the loop, evaluate the polynomial at the best position: y* = poly(closest_x).
15 Define the maximum camber point: max_pt = (int(closest_x), int(y*)).
16 **Return** max_pt.

Listing 19 - Calculation of Maximum Camber Point.

Listing 19 finds the exact point of maximum camber along a fitted polynomial curve of the sail line. It first calculates the slope of the chord and defines a search interval along the x-axis between the chord's endpoints. The interval is divided into a fixed number of sections,

and for each position, the tangent slope of the polynomial is compared to the chord slope. The point where this mismatch is smallest is chosen as the best candidate. After evaluating the polynomial at this position, the coordinates of the maximum camber point are obtained and returned.

These aerodynamic parameters are calculated independently for the top, middle and bottom lines, and yield a set of three values for each parameter for each frame.

For verification and analysis, the processed frames are annotated with visual markers. They also have chord endpoints, maximum camber points, and line highlights, allowing the segmentation quality to be visually assessed as seen in Figure 24. The sail lines are colour-coded for an intuitive display of the segmentation results in the original image, as seen in Listing 20.

Algorithm 20: Draw Cross Markers for Chord Points and Max Camber

Input: A frame image (overlay) where results will be displayed and a list of section properties (props) that contains the chord points and the maximum camber point for each detected section.
Output: The frame image displayed with crosses marking the chord endpoints and the maximum camber point.

```
1  Create a drawable copy of the frame: disp = overlay.copy().
2  FOR each section sec in props:
3      Draw a cross at the chord start (sec["first"].x, sec["first"].y).
4      Draw a cross at the chord end (sec["last"].x, sec["last"].y).
5      Draw a cross at the maximum camber (sec["max_pt"].x, sec["max_pt"].y).
6  Show the preview window with the image disp.
```

Listing 20 - Draw Cross Markers for Chord Points and Max Camber.

The algorithm above adds visual markers to highlight important geometric points on a frame. A copy of the original overlay image is made for safe drawing. For each detected section, crosses are drawn at the starting point of the chord, at its ending point, and at the maximum camber point. These markers help to confirm the accuracy of the computed properties visually. Once all sections are marked, the updated frame appears in a preview window (Figure 24).

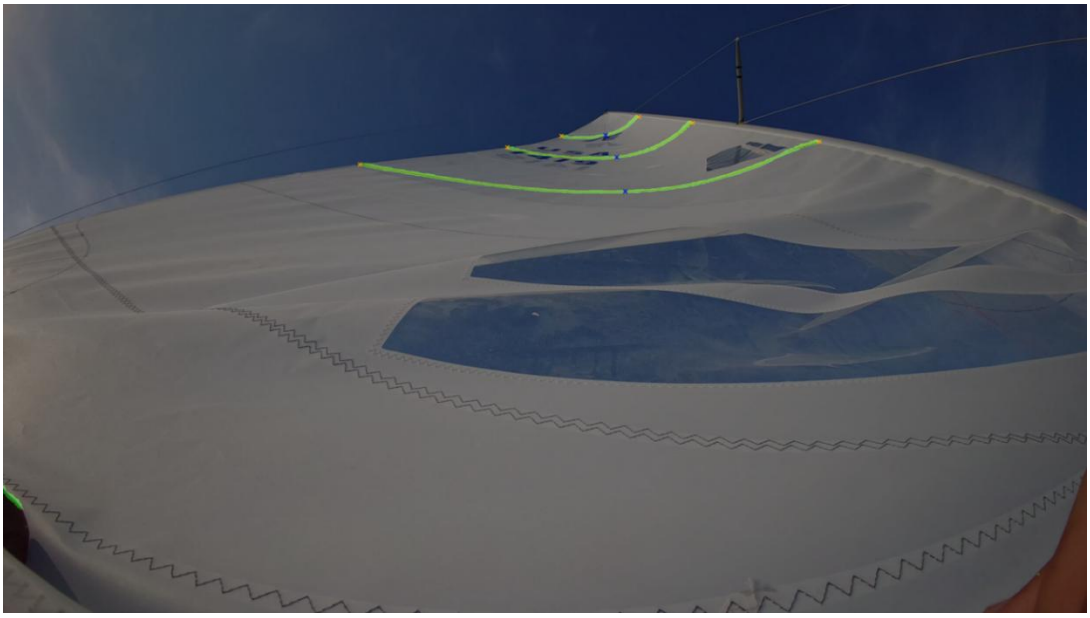


Figure 24 - Visual Markers of Chord Points and Mask Segmentation.

All the calculated parameters are stored in a formatted CSV file, one row per frame and one column per parameter-section combination (e.g., Camber_Top, Draft_Mid, Twist_Bot).

This design allows for easy interfacing with other analysis tools and statistical measures for later comparisons. Furthermore, the three crucial aerodynamic parameter values are printed, per frame, into the console during execution for immediate inspection of results, as seen in Figure 25.

	Camber	Draft	Twist
Top	12.75	46.52	15.99
Mid	12.93	42.52	8.28
Bot	9.31	42.94	3.40

Figure 25 – Example of Aerodynamic Parameters Text Interface.

The code implementation for this text interface is demonstrated in Listing 21:

Algorithm 21: Display Per-Frame Metrics Table (Top / Mid / Bot).

Input: A list of section properties (props) that may contain up to three sections, each with values for camber, draft, and twist, together with the predefined section names ["Top", "Mid", "Bot"].
Output: A printed table where the rows correspond to Top, Mid, and Bot, and the columns correspond to Camber, Draft, and Twist, with all values rounded to two decimals, and a CSV file storing these results for all frames.

```
1  Initialise empty lists: c_vals, d_vals, t_vals.
2  FOR i = 0 TO 2:
3      IF i < len(props) THEN
4          Set sec = props[i].
5          Append round(sec["camber"], 2) to c_vals.
6          Append round(sec["draft"], 2) to d_vals.
7          Append round(sec["twist"], 2) to t_vals.
8      ELSE
9          Append 0.00 to c_vals, d_vals, and t_vals.
   Build a table with:
10     Rows = section names (Top, Mid, Bot).
11     Columns = Camber, Draft, Twist.
12     Values from c_vals, d_vals, t_vals.
13 Print the table as text with two-decimal formatting.
14 Store the results across all frames into a CSV file for later analysis.
```

Listing 21 - Display Per-Frame Metrics Table (Top / Mid / Bot).

Listing 21 organises the aerodynamic properties of detected sail sections into a structured table for analysis. For each of the three predefined sections, Top, Mid, and Bottom, it collects the values of camber, draft, and twist. If any section is missing, it inserts zeros to keep the table complete. The data is assembled into a table with rows representing the three sail sections and columns showing the aerodynamic metrics. The table displays values rounded to two decimals for readability. The results are also saved in a CSV file, which helps track the metrics consistently across all processed frames.

6 Comparative Evaluation

This chapter compares the SegFormer deep learning, and a traditional image processing method applied to line detection and sail parameter calculations. The comparison considers both the output quality from the segmentation and the algorithm's computational efficiency. The study uses a set of twenty-three videos examined under identical conditions, enabling a balanced comparison of the two approaches. Each video's orange lines and aerodynamic parameters — Camber, Draft, and Twist — were compared against manually-annotated ground-truth values to compute each approach's accuracy. In parallel, the time it took to process each video was monitored to evaluate the algorithm's computational efficiency for both techniques. This analysis makes it possible to understand the trade-offs between detection quality and computation speed and gain insight into the suitability for different operational environments, such as offline performance assessment or real-time onboard systems.

6.1 Hardware Configuration

All experiments and system development occurred on a consumer-grade computer equipped with a GPU. Although the computational costs of deep learning training and high-resolution image processing were high, the hardware could push the entire pipeline at a sufficiently efficient rate. This displays the potential for deployment of the system on accessible, non-specialised machines. The hardware configuration used during the project is presented in Table 2.

Table 2 - Hardware Specification.

Component	Specification
Laptop	ASUS TUF Dash F15 (FX516PM)
CPU	Intel Core i7-11370H, 4 cores / 8 threads @ 3.3 GHz
GPU	NVIDIA RTX 3060 Laptop GPU, 6 GB VRAM
RAM	16 GB DDR4
OS	Windows 11 Home Version 24H2

6.2 Methodology for Comparison

The testing was performed by executing the SegFormer model and the traditional image processing method in the same Python environment for the sake of fairness when comparing the performance of the two models. The complete pipeline was executed in both

instances — from input videos to detecting the orange lines, computation of the aerodynamic parameters, and final output storage.

Written initially in C++, the traditional algorithm was adapted to Python to run under the same environment as SegFormer. It uses conventional CV operations such as edge detection, contour analysis, and geometric measurement for sail line detection and calculations of the aerodynamic parameters. The SegFormer model, on the other hand, uses a transformer-based semantic framework for line segmentation and geometric post-processing to calculate the same parameters.

Both approaches compared results with ground-truth measurements gathered through manual annotation, using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Pearson's correlation coefficient. In addition, the win-rate percentage was also computed per video to determine which method led to parameter values closest to the ground truth.

6.3 Training and Validation Metrics

The training logs in Figure 26 show that the model was trained for 30 epochs. During the initial epochs, all metrics improved steadily as the model learned to recognise line patterns. However, after **around six epochs**, performance values stabilised and remained consistent for the remainder of training.

- Global validation accuracy stabilised around 99.4%.
- Line-specific accuracy validation remained between 85–86%.
- IoU for sail line pixels stagnated around 0.71–0.72 in the validation phase.
- Dice coefficient converged to values between 0.83 and 0.84 in the validation phase.

```
=== Epoch 15/30 – LR 2.50e-05 – 570.8s – PeakMem 1.30 GB ===
 Train | Loss 0.1015 | Acc 0.9962 | IoU[line]=0.7827 | Dice[line]=0.8781 | Acc[line]=0.9079
 Val   | Loss 0.1391 | Acc 0.9947 | IoU[line]=0.7140 | Dice[line]=0.8332 | Acc[line]=0.8472

=== Epoch 16/30 – LR 2.50e-05 – 585.4s – PeakMem 1.30 GB ===
 Train | Loss 0.1032 | Acc 0.9963 | IoU[line]=0.7816 | Dice[line]=0.8774 | Acc[line]=0.9069
 Val   | Loss 0.1381 | Acc 0.9947 | IoU[line]=0.7151 | Dice[line]=0.8339 | Acc[line]=0.8583

=== Epoch 17/30 – LR 2.50e-05 – 589.1s – PeakMem 1.30 GB ===
 Train | Loss 0.1031 | Acc 0.9962 | IoU[line]=0.7810 | Dice[line]=0.8770 | Acc[line]=0.9046
 Val   | Loss 0.1391 | Acc 0.9947 | IoU[line]=0.7143 | Dice[line]=0.8334 | Acc[line]=0.8553

=== Epoch 18/30 – LR 1.25e-05 – 577.6s – PeakMem 1.30 GB ===
 Train | Loss 0.1040 | Acc 0.9963 | IoU[line]=0.7798 | Dice[line]=0.8763 | Acc[line]=0.9040
 Val   | Loss 0.1373 | Acc 0.9947 | IoU[line]=0.7164 | Dice[line]=0.8348 | Acc[line]=0.8635

=== Epoch 19/30 – LR 1.25e-05 – 589.4s – PeakMem 1.30 GB ===
 Train | Loss 0.1012 | Acc 0.9963 | IoU[line]=0.7841 | Dice[line]=0.8790 | Acc[line]=0.9099
 Val   | Loss 0.1387 | Acc 0.9947 | IoU[line]=0.7137 | Dice[line]=0.8329 | Acc[line]=0.8467

=== Epoch 20/30 – LR 1.25e-05 – 597.2s – PeakMem 1.30 GB ===
 Train | Loss 0.1004 | Acc 0.9963 | IoU[line]=0.7824 | Dice[line]=0.8779 | Acc[line]=0.9059
 Val   | Loss 0.1356 | Acc 0.9947 | IoU[line]=0.7174 | Dice[line]=0.8355 | Acc[line]=0.8660
```

Figure 26 - Training Logs for SegFormer Mit-B1.

This behaviour indicates that the model converged early and had stable performance without drastic overfitting or loss of performance in the later epochs. While the overall accuracy is very high, the line-specific measurements provide a more realistic estimate of the model's ability to segment thin structures and confirm that the architecture is effective but limited by the very small percentage of line pixels in the dataset.

Training was also conducted using SegFormer MiT-B2 and MiT-B3 backbones to verify the model's robustness further. The results were **very similar** to those of MiT-B1, with the same early convergence and stable outcomes for the whole range of the accuracy, IoU, and Dice metric. This consistency across multiple backbone variants indicates that the performance is not dependent on model size within the MiT family. This strengthens the conclusion that the dataset and task characteristics dominate the achievable performance more than incremental architectural variations.

6.4 Processing Time Analysis

Processing time is an important factor when looking at how well a detection system works, especially for real-time or nearly real-time applications. In this study, the processing time was measured only during the inference stage of the models, leaving out video decoding and file saving tasks. It repeated each experiment 100 times for every one of the 23 videos in the dataset, and sampled exactly 20 frames from each one. This approach followed the methods used in earlier studies to ensure the same frames were used across all methods. This allowed for a fair comparison of both accuracy and efficiency.

The results in Table 3 show a noticeable difference in performance between the methods. The traditional image processing pipeline consistently had much shorter times per frame, while the SegFormer model took significantly longer for inference, especially as the input resolution increased.

Additionally, the difference in processing time depends not on the video length but on the **frame resolution**. For example, the first three videos, which were in 3840×2160 (4K), needed much more processing time than Video 4, which was in 1920×1080 (Full HD). This difference happens in the case of SegFormer because each frame is broken down into several tiles with a resolution of 512×512 pixels for processing. Thus:

- In a **1080p (1920×1080)** video, a frame generates about **12 tiles**,
- In a **4K (3840×2160)** video, a frame generates about **40 tiles**,
- In a **5K (5312×2988)** video, a frame generates about **66 tiles**.

Equation 5 follows directly from dividing the frame dimensions by the tile size. For example, in a 1080p frame:

$$tiles\ in\ width = \left\lceil \frac{1920}{512} \right\rceil = 4, tiles\ in\ height = \left\lceil \frac{1080}{512} \right\rceil = 3$$

$$4 \times 3 = 12\ tiles\ per\ frame \quad \text{Eq. (5)}$$

Equation 5 - Tiles per frame dimensions.

The same reasoning applies to higher resolutions. When the frame size increases, the number of tiles also increases. This leads to more inference passes being required. As a result, the processing time grows with the resolution of the frames.

In contrast, the traditional method is much less affected by resolution. Although processing times increase with more pixels, the tasks it performs — like thresholding, morphology, contour tracing, and basic geometric calculations — are simple and adjust more smoothly with image size. For instance, **Video 1 (4K) took 0.075 seconds per frame**, while **Video 4 (1080p) took just 0.024 seconds**. However, in **SegFormer mit-b1**, the difference was much bigger: **1.256 seconds per frame for Video 1** compared to **0.378 seconds for Video 4**. On average, SegFormer mit-b1 was around **15x slower** than the traditional approach, with mit-b2 and mit-b3 being even slower. This shows how high-resolution inputs impact transformer-based models more significantly due to their tiling strategy, while the conventional method stays efficient even with larger sizes.

Additional experiments with other SegFormer backbones, mit-b2 and mit-b3, increased inference time. In some cases, the runtime more than doubled compared to mit-b1, but there were no significant improvements in seam detection accuracy.

Table 3 - Processing Time Comparison per-frame in seconds.

Video	Traditional (a)	SegFormer MiT-B1 (b)	SegFormer MiT-B2 (c)	SegFormer MiT-B3 (d)	Size of the video
1	0.075	1.256	2.912	4.048	3840 × 2160
2	0.074	1.261	2.901	4.049	3840 × 2160
3	0.074	1.258	2.900	4.050	3840 × 2160
4	0.024	0.378	0.876	1.206	1920 × 1080
5	0.074	1.258	2.908	4.042	3840 × 2160
6	0.023	0.377	0.874	1.212	1920 × 1080
7	0.074	1.256	2.914	4.051	3840 × 2160
8	0.074	1.254	2.909	4.043	3840 × 2160
9	0.073	1.257	2.912	4.049	3840 × 2160
10	0.138	2.082	4.798	6.676	5312 × 2988
11	0.023	0.375	0.875	1.211	1920 × 1080
12	0.040	0.571	1.318	1.824	2704 × 1520
13	0.075	1.261	2.918	4.048	3840 × 2160
14	0.040	0.572	1.307	1.822	2704 × 1520

15	0.075	1.259	2.910	4.073	3840 × 2160
16	0.024	0.377	0.878	1.223	1920 × 1080
17	0.076	1.260	2.907	4.088	3840 × 2160
18	0.076	1.255	2.912	4.061	3840 × 2160
19	0.023	0.377	0.882	1.213	1920 × 1080
20	0.024	0.378	0.883	1.211	1920 × 1080
21	0.139	2.083	4.801	6.675	5312 × 2988
22	0.024	0.378	0.873	1.210	1920 × 1080
23	0.024	0.378	0.878	1.219	1920 × 1080

The analysis shows that traditional image processing techniques are the most efficient solution when processing speed is the main requirement. They consistently deliver much faster results than transformer-based approaches. Among the SegFormer backbones tested, mit-b1 provided the best balance between accuracy and runtime. In contrast, mit-b2 and mit-b3 added unnecessary computational load without offering meaningful benefits. As a result, mit-b1 was identified as the most suitable backbone for this thesis's goals.

6.5 Analysis of Segmentation Quality and Parameter Computation

Segmentation accuracy directly impacts the validity of aerodynamic parameters inferred for detected lines. To quantify accuracy in this thesis, comparisons between Segformer outputs and traditional methods' results were performed against manually annotated ground-truth masks and parameter values.

The ground truth was obtained through a manual annotation process. The sail lines were manually outlined for each test frame to generate binary segmentation masks, ensuring every visible line was represented correctly, even in challenging scenarios such as shadows or deformations. Further, chord points were set manually along annotated lines in such a way as to mark the leading/trailing edges of observed sail panels. Such chord endpoints were employed as geometric reference for aerodynamic calculations. Also, fitting a polynomial curve over manually delineated lines and picking maximum camber points along a chord permitted estimation of reference values for camber, draft, and twist. Such an approach ensured that not only were pixel-level masks for sail lines included within a ground-truth dataset, but even parameter estimations were made available with direct human inspection.

To test reliability, a win-rate experiment was performed. Under this method, each prediction by Segformer and the traditional method is compared against the manually annotated ground-truth parameter values. For each frame for each parameter (camber, draft, twist), an algorithm whose estimate is numerically closest to ground truth is considered a "winner"

for that case. The final win-rate is then determined by counting up wins over all frames over all video recordings.

Furthermore, Table 4, Table 5 and Table 6 compare the SegFormer variants (MiT-B1, MiT-B2, MiT-B3) and the traditional image processing technique based on segmentation accuracy. Each table summarises all test videos with an accuracy report specific to sail lines and error measures such as MAE and RMSE.

The quantitative results also confirmed this advantage. SegFormer had lower values for its MAE and RMSE in Camber, Draft, and Twist while maintaining correlation scores that were significantly greater when compared to the ground truth values.

Table 4 - Traditional Technique vs SegFormer Mit-B1 in Segmentation Accuracy.

Video	SegFormer Accuracy (%)	Traditional Accuracy (%)	MAE (SegFormer)	RMSE (SegFormer)	R (SegFormer)	MAE (Traditional)	RMSE (Traditional)	R (Traditional)	Preferred Approach (Lower Error vs. Ground Truth)
1	60.89	39.11	1.00	3.04	1.00	4.73	7.98	0.99	SegFormer
2	100.00	0.00	0.00	0.00	1.00	0.89	1.55	1.00	SegFormer
3	54.49	45.51	0.76	1.61	1.00	1.61	8.41	0.85	SegFormer
4	43.58	56.42	1.25	5.52	0.94	1.45	5.78	0.93	Traditional
5	46.70	53.30	0.83	3.90	1.00	3.91	9.49	0.88	Traditional
6	57.78	42.22	0.43	0.54	1.00	0.64	1.45	1.00	SegFormer
7	50.00	50.00	0.82	16.87	0.76	0.76	15.54	0.78	Tie
8	60.89	39.11	4.89	9.24	0.92	3.68	10.48	0.86	SegFormer
9	71.91	28.09	6.33	15.00	0.77	10.85	16.67	0.71	SegFormer
10	71.67	28.33	1.13	2.43	0.99	6.46	11.67	0.90	SegFormer
11	51.97	48.03	0.72	0.93	1.00	0.69	0.97	1.00	SegFormer
12	45.98	54.02	0.92	1.68	1.00	0.68	1.97	0.99	Traditional
13	51.96	48.04	0.49	0.89	1.00	0.55	1.12	1.00	SegFormer
14	58.76	41.24	0.37	0.87	1.00	0.97	2.32	0.99	SegFormer
15	41.81	58.19	0.38	0.55	1.00	0.29	0.44	1.00	Traditional
16	66.10	33.90	0.36	0.63	1.00	0.35	0.81	0.98	SegFormer
17	53.98	46.02	0.38	0.65	1.00	0.42	0.51	1.00	SegFormer
18	48.02	51.98	0.38	0.49	1.00	0.34	0.46	1.00	Traditional
19	60.00	40.00	0.56	0.76	1.00	0.62	0.73	1.00	Traditional
20	49.72	50.28	0.44	0.78	1.00	0.52	0.73	1.00	Traditional
21	67.22	32.78	1.61	7.78	0.93	7.11	12.57	0.89	SegFormer
22	46.89	53.11	0.61	0.85	1.00	0.51	0.65	1.00	Traditional
23	56.25	43.75	0.41	0.59	1.00	1.14	2.55	0.99	SegFormer

Table 5 - Traditional Technique vs SegFormer Mit-B2 in Segmentation Accuracy.

Video	SegFormer Accuracy (%)	Traditional Accuracy (%)	MAE (SegFormer)	RMSE (SegFormer)	R (SegFormer)	MAE (Traditional)	RMSE (Traditional)	R (Traditional)	Preferred Approach (Lower Error vs. Ground Truth)
1	88.89	11.11	0.49	3.04	1.00	4.73	7.98	0.99	SegFormer
2	82.02	17.98	0.27	0.61	1.00	0.89	1.55	1.00	SegFormer
3	53.41	46.59	0.91	3.84	0.97	1.61	8.41	0.85	SegFormer
4	54.44	45.56	0.98	4.23	0.97	1.45	5.78	0.93	SegFormer

5	50.28	49.72	1.33	8.32	0.91	3.91	9.49	0.88	SegFormer
6	58.99	41.01	0.42	0.54	1.00	0.64	1.45	1.00	SegFormer
7	46.97	53.93	0.76	15.61	0.79	0.76	15.54	0.78	Traditional
8	57.39	42.61	4.67	9.92	0.87	3.68	10.48	0.86	SegFormer
9	61.24	38.76	6.03	15.80	0.73	10.85	16.67	0.71	SegFormer
10	71.35	28.65	1.20	4.20	0.98	6.46	11.67	0.90	SegFormer
11	53.37	46.63	0.70	0.98	1.00	0.69	0.97	1.00	SegFormer
12	41.43	58.57	0.86	1.72	1.00	0.68	0.97	0.99	Traditional
13	51.96	48.04	0.65	2.02	0.99	0.55	1.12	1.00	SegFormer
14	52.54	47.46	0.72	1.62	1.00	0.97	2.32	0.99	SegFormer
15	41.01	58.99	0.39	0.56	0.56	0.29	0.44	1.00	Traditional
16	58.76	41.24	0.69	0.85	1.00	3.10	3.10	0.99	SegFormer
17	52.51	47.49	0.38	0.68	1.00	0.42	0.51	1.00	SegFormer
18	37.29	62.71	0.44	0.55	1.00	0.34	0.46	1.00	Traditional
19	69.83	30.17	0.17	0.63	1.00	0.62	0.73	1.00	SegFormer
20	53.98	46.02	0.46	0.73	1.00	0.52	0.73	1.00	SegFormer
21	76.97	23.03	0.60	1.63	1.00	7.11	12.57	0.89	SegFormer
22	48.88	51.12	0.59	0.77	1.00	0.51	0.65	1.00	Traditional
23	58.10	41.90	0.41	0.60	1.00	1.14	2.55	0.99	SegFormer

Table 6 - Traditional Technique vs SegFormer Mit-B3 in Segmentation Accuracy.

Video	SegFormer Accuracy (%)	Traditional Accuracy (%)	MAE (SegFormer)	RMSE (SegFormer)	R (SegFormer)	MAE (Traditional)	RMSE (Traditional)	R (Traditional)	Preferred Approach (Lower Error vs. Ground Truth)
1	88.89	11.11	0.34	0.49	1.00	4.73	7.98	0.99	SegFormer
2	83.71	16.29	0.32	0.79	1.00	0.89	1.55	1.00	SegFormer
3	60.89	39.11	0.49	0.70	1.00	1.61	8.41	0.85	SegFormer
4	48.02	51.98	0.67	1.56	0.97	1.45	5.78	0.93	Traditional
5	43.26	56.74	0.86	1.31	0.91	3.91	9.49	0.88	Traditional
6	41.95	58.05	0.57	0.70	1.00	0.64	1.45	1.00	Traditional
7	57.06	42.94	0.62	0.75	0.79	0.76	15.54	0.78	SegFormer
8	59.78	40.22	4.27	9.52	0.88	3.68	10.48	0.86	SegFormer
9	67.60	32.40	7.53	15.86	0.73	10.85	16.67	0.71	SegFormer
10	76.54	23.46	0.54	0.83	1.00	6.46	11.67	0.90	SegFormer
11	56.67	43.33	0.66	0.86	1.00	0.69	0.97	1.00	SegFormer
12	50.85	49.15	0.51	0.92	0.99	0.68	0.97	0.99	Traditional
13	51.96	48.04	0.65	2.02	0.99	0.55	1.12	1.00	SegFormer
14	50.85	49.15	0.41	0.65	1.00	0.97	2.32	0.99	SegFormer
15	39.66	60.34	0.44	0.65	0.56	0.29	0.44	1.00	Traditional
16	59.32	40.68	0.36	0.59	1.00	3.10	3.10	0.99	SegFormer
17	43.75	56.25	0.43	0.53	1.00	0.42	0.51	1.00	SegFormer
18	39.55	60.45	0.44	0.56	1.00	0.34	0.46	1.00	Traditional
19	43.21	56.79	0.49	0.67	1.00	0.62	0.73	1.00	SegFormer
20	41.81	58.19	0.59	0.71	1.00	0.52	0.73	1.00	Traditional
21	59.13	40.87	0.73	1.03	0.72	7.11	12.57	0.89	SegFormer
22	45.45	54.55	0.63	0.81	1.00	0.51	0.65	1.00	Traditional
23	57.39	42.61	0.43	0.63	1.00	1.14	2.55	0.99	SegFormer

The comparison in Table 4, Table 5 and Table 6 shows that while SegFormer generally achieves higher segmentation accuracy than the conventional method, there are several videos (like 4, 5, 12, 15, 18, 20, and 22) where its performance drops. In these cases, the

traditional approach performs equally well or better. The specific sailing conditions in those videos can explain this behaviour.

When the sail is **luffing**, it flaps due to unstable airflow, so the orange contour line becomes highly irregular and unstable. Instead of forming a smooth, continuous edge, the sail line waves. This creates rapid deformations, motion blur, and fragmented boundaries. Since SegFormer learns consistent patterns from training data, these unstable shapes create ambiguity, making it hard for the model to maintain accurate segmentation.

In some frames, the orange contour lines are partially **covered by letters or markings on the sail**, such as the boat's name or ID. This creates confusion for the model. In other cases, the orange lines are so thin in the video frames that they take up only a few pixels, making it difficult for SegFormer to detect them.

Therefore, the drops in accuracy seen in certain videos are not failures of the model's architecture. They result from challenges like luffing, sail lettering, and very thin contour lines, which remain difficult even for state-of-the-art deep learning models.

Additionally, win-rate analysis found that in most videos, estimations by Segformer were closer to the ground-truth values than the traditional method, as shown in Table 7.

Table 7 - Win-rate analysis of SegFormer Variants vs Traditional Technique.

SegFormer Variant	SegFormer achieved lower error	Traditional achieved lower error	Ties	Overall Best Approach
MIT-B1	2347	1751	42	SegFormer
MIT-B2	2350	1740	50	SegFormer
MIT-B3	2285	1801	54	SegFormer

Figure 27 (a) presents the results of the traditional image processing approach. It can detect certain portions of green over orange lines, but cannot provide continuous and precise localisation for the entire sail area. The red circles indicate regions where detection failure is observed: not only are lines ripped, but at times entirely missed or misplaced in the background texture. Failures are most evident at the top of the sail, where irregular lighting, its own reflection in the fabric, and deformation by the wind create noise for the algorithm to detect the orange sail lines.

Compared to Figure 27 (b), in which the segmentation predicted by the SegFormer model is shown, Figure 27 (a) shows that the traditional methodology is less robust. Unlike the AI-based method, which achieves higher continuity and accuracy even in trouble-prone environments, traditional methodology is characterised by disrupted line segmentation and ignored areas. These frailties bear witness to environmental variability inherent in hand-designed, traditional-based algorithms, to limit their use in aerodynamic parameter extraction.

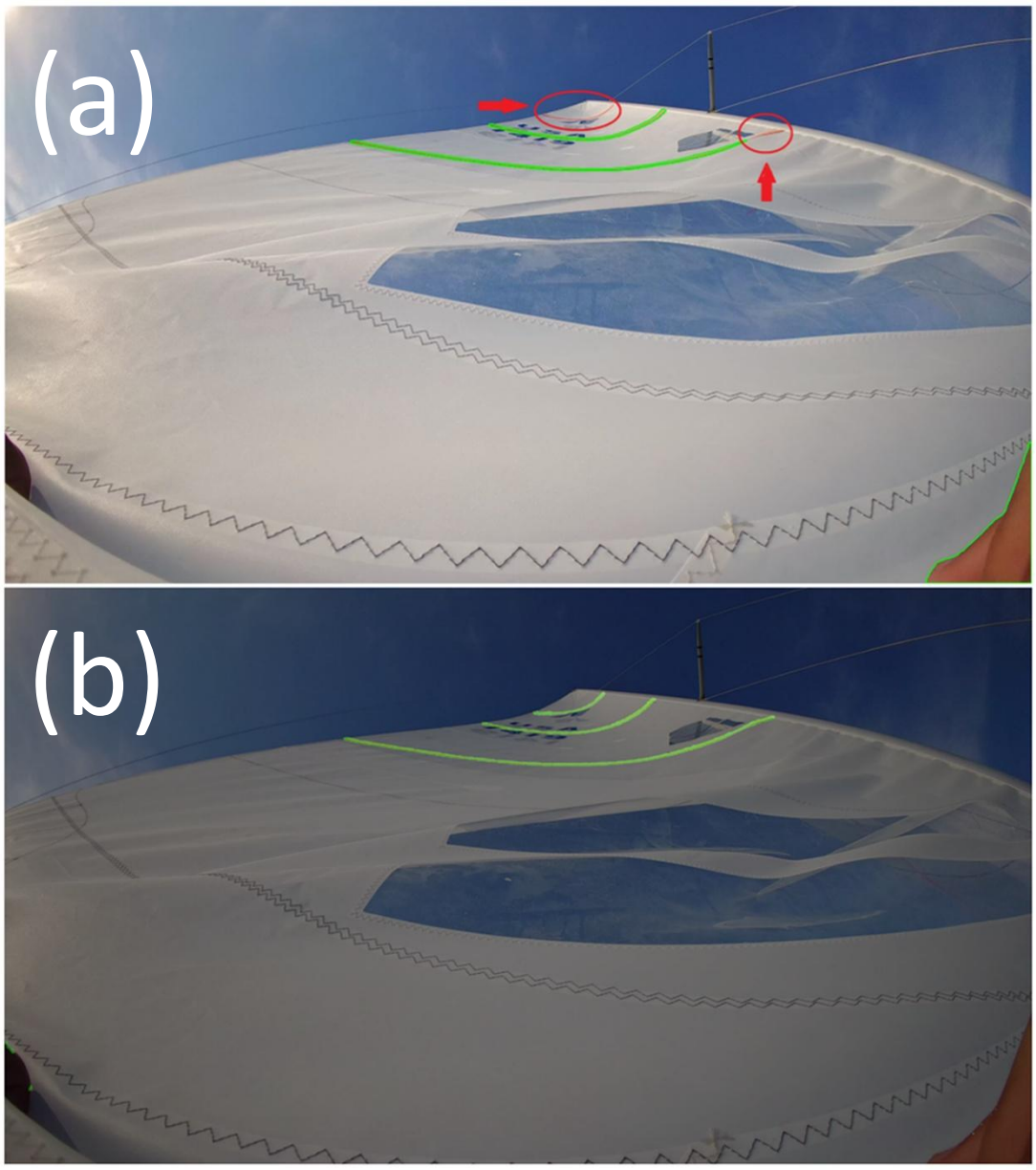


Figure 27 - Comparison of sail line detection between (a) traditional image processing technique and (b) SegFormer.

The latter experiment with the mit-b2 and mit-b3 backbones provided valuable insight into the importance of model performance. It increased the inference time but decreased segmentation accuracy, mostly in fine detail segmentation in the lines of a sailboat, which directly impacts aerodynamic calculations. It further confirmed the selection of the mit-b1 backbone as the best configuration for this application, balancing accuracy and speed.

7 Conclusions

This thesis offered a complete pipeline to extract aerodynamic parameters — twist, draft, and camber — from sail footage using semantic segmentation. The proposed solution combined a deep learning-based approach using Segformer with geometrical post-processing techniques to calculate accurate sail shape parameters. The system was compared to a traditional image processing method based on thresholding and morphological processing using OpenCV. Even with faster processing times, it did not capture full line contours in most instances — approximately 70% of frames — especially under varying lighting, deformation, or partial blockage.

The SegFormer model, on the other hand, produced better, smoother segmentation outputs, thereby providing much better aerodynamic estimations. Three SegFormer backbones — MiT-B1, MiT-B2, and MiT-B3 — were tested. Although MiT-B2 and MiT-B3 offered similar accuracy, they introduced a higher computational cost and did not improve fine sail line detail segmentation. In the end, MiT-B1 was the optimal setting, achieving a good balance between accuracy, smooth continuity of lines, and inference time. Despite SegFormer taking longer to compute than the traditional approach, the enhanced segmentation quality and parameter accuracy made up for the computational cost. A win-rate analysis supported the comparison, as Segformer produced parameter values closer to the ground truth in most frames, confirming its overall superiority in all the test videos. The traditional algorithm is still beneficial when latency is key and resources are limited. The AI-based approach, particularly SegFormer mit-b1, is a more accurate option when sufficient computational resources are available. These results show that integrating AI-powered computer vision into sailing performance analysis is possible. This integration allows for the exact and automatic extraction of aerodynamic parameters, directly supporting optimisation and decision-making in real sailing conditions.

7.1 Future Work

The future development of this system should focus on transforming this offline, post-processing program into a fully integrated, real-time application capable of independently operating on embedded or portable hardware during sailing sessions.

Another important option is executing the whole pipeline in real-time while the camera is actively recording. The goal is to process every video frame by the algorithm, separate sail seams in real-time, and display the estimated aerodynamic parameters — camber, draft, and twist — the whole time on the screen of a synchronised device, such as a tablet or onboard display, during sailing sessions. This will give sailors immediate visual and quantitative feedback about sail trim and shape on the go.

To provide such capability, the system must go through extensive optimisation, including segmentation inference acceleration, parallel processing to enable UI updates and logging, and interaction with the camera and display hardware. All these optimisations will be critically important in allowing smooth, real-time operation without drop frames or lag.

At the same time, the dataset used for training should be expanded by capturing more sail types, fabrics, colours, and lighting, reflection, and motion blur situations to enhance model robustness and facilitate broader application to different sailboats.

Another primary goal is using embedded systems such as the NVIDIA Jetson Nano. These systems offer good tradeoffs between energy efficiency and computational power, perfect for onboard deployments. Training lighter models natively on the desired hardware or porting the current pipeline with tools such as PyTorch Mobile will help to reach compatibility and stable operation in constrained environments.

Lastly, it will be essential to generalise the seam detection and parameter computation to more complex sail geometries and camera perspectives.

Together, these future developments would change the current system into a powerful, real-time sail analysis tool that provides in-session insights — all very valuable to sailors, coaches, and sail designers.

8 References

[1] "Cross-training camp delivers training success for VIS 'foilers' — Victorian Institute of Sport." Accessed: Dec. 20, 2024. [Online]. Available: https://vis.org.au/news/2023/09/cross-training-camp-delivers-training-success-for-vis-foilers

[2] G. Lopes, "Sailshape," Instituto Superior de Engenharia do Porto, Porto, 2024.

[3] "Understanding Sailboats and Sailing - The Sails." Accessed: Dec. 20, 2024. [Online]. Available: https://asa.com/news/2022/02/22/understanding-sailboats-the-sails/

[4] "Shaping your sail with Draft - Twist - Angle of Attack - Camber - Wind Conditions - Mandurah Yacht Academy." Accessed: Dec. 20, 2024. [Online]. Available: https://mandurahyachtacademy.au/shaping-your-sail-with-draft-twist-angle-of-attack-camber-wind-conditions/

[5] "Shaping Your Mainsail, Part 2: Camber - SailZing.com." Accessed: Dec. 20, 2024. [Online]. Available: https://sailzing.com/shaping-your-mainsail-part-2-camber/

[6] "Shaping Your Mainsail, Part 3: Draft Shape and Position - SailZing.com." Accessed: Dec. 20, 2024. [Online]. Available: https://sailzing.com/shaping-your-mainsail-part-3-draft-shape-and-position/

[7] "Shaping Your Mainsail, Part 4 - Controlling Twist - SailZing.com." Accessed: Dec. 20, 2024. [Online]. Available: https://sailzing.com/shaping-your-sail-part-4-controlling-twist/

[8] J. A. Carranza, S. Zaidi, and H. Carranza, "Plane Detection Based Object Recognition for Augmented Reality," *academia.edu*, Accessed: Jan. 05, 2025. [Online]. Available: https://www.academia.edu/download/87612944/CDSR_305.pdf

[9] V. Jindal, S. N. Singh, S. S. K.-M. I. and Data, and undefined 2022, "Facial Recognition with Computer Vision," *Springer*, Accessed: Jan. 05, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-19-2347-0_24

[10] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: A survey," *ieeexplore.ieee.orgDK Prasad, D Rajan, L Rachmawati, E Rajabally, C QuekIEEE Transactions on Intelligent Transportation Systems, 2017•ieeexplore.ieee.org*, Accessed: Jan. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7812788/

[11] E. Wisernig, T. Sadhu, C. Zilinski, B. Wyvill, A. B. Albu, and M. Hoeberechts, "Augmented Reality Visualization for Sailboats (ARVS)," *Proceedings - 2015*

International Conference on Cyberworlds, CW 2015, pp. 61–68, Feb. 2016, doi: 10.1109/CW.2015.74.

[12] Z.-Y. Huang *et al.*, "A study on computer vision for facial emotion recognition," *nature.com*, 123AD, doi: 10.1038/s41598-023-35446-4.

[13] V. Mosiichuk, "Deep Learning for Automated Adequacy Assessment of Cervical Cytology Samples," 2022, Accessed: Jan. 05, 2025. [Online]. Available: https://search.proquest.com/openview/614b5e06d6d11924f6de613d667c6bf6/1?pq-origsite=gscholar&cbl=2026366&diss=y

[14] J. Janai, F. Güney, A. Behl, … A. G. T. in C., and undefined 2020, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *nowpublishers.com*, Accessed: Jan. 05, 2025. [Online]. Available: https://www.nowpublishers.com/article/Details/CGV-079

[15] H. Pan *et al.*, "Application of Computer Vision Technology in Industrial Automation," *iopscience.iop.org*, vol. 2037, p. 12015, 2021, doi: 10.1088/1742-6596/2037/1/012015.

[16] A. Khan, A. Laghari, S. A.-E. E. T. on, and undefined 2021, "Machine learning in computer vision: a review," *publications.eai.eu*, 2021, doi: 10.4108/eai.21-4-2021.169418.

[17] G. Astolfi *et al.*, "Syntactic pattern recognition in computer vision: A systematic review," *dl.acm.org*, vol. 54, no. 3, Apr. 2021, doi: 10.1145/3447241.

[18] V. Hrytsyk and M. Nazarkevych, "Real-Time Sensing, Reasoning and Adaptation for Computer Vision Systems," *Lecture Notes on Data Engineering and Communications Technologies*, vol. 77, pp. 573–585, 2022, doi: 10.1007/978-3-030-82014-5_39.

[19] L. Zhou, L. Zhang, N. K.-I. T. on Systems, undefined Man, and undefined 2022, "Computer vision techniques in manufacturing," *ieeexplore.ieee.org*, Dec. 2021, doi: 10.36227/techrxiv.17125652.v1.

[20] S. Kaushal, D. Tammineni, P. Rana, … M. S.-T. in F. S., and undefined 2024, "Computer vision and deep learning-based approaches for detection of food nutrients/nutrition: New insights and advances," *Elsevier*, Accessed: Jan. 05, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924224424000840?casa_token=arFlV2FStIQAAAAA:kKaFlW2HF47z1D2R1t2MsLCtGI4-dqaLZDPEaupeRxIxxY4ujgVPXbTh9YZ5rDb3d6Bj8wdCCw

[21] K. Bjerge, H. M. R. Mann, and T. T. Høye, "Real-time insect tracking and monitoring with computer vision and deep learning," *Remote Sens Ecol Conserv*, vol. 8, no. 3, pp. 315–327, Jun. 2022, doi: 10.1002/RSE2.245.

[22] R. Kumar, R. Ramya, M. Balaji, … V. H.-2024 I., and undefined 2024, “Garbage Collection and Segregation using Computer Vision,” *ieeexplore.ieee.org*, Accessed: Jan. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10544861/

[23] P. Peiro, C. Q. Gómez Muñoz, and F. P. GarcíaMárquez, “Use of UAVS, Computer Vision, and IOT for Traffic Analysis,” *International Series in Operations Research and Management Science*, vol. 305, pp. 275–296, 2021, doi: 10.1007/978-3-030-70478-0_13.

[24] I. Gupta and D. P. Kaur, “FPGA based Feature Extraction in Real time Computer Vision-A Comprehensive Survey,” *2022 International Conference on Signal and Information Processing, IConSIP 2022*, 2022, doi: 10.1109/ICONSIP49665.2022.10007508.

[25] “What is feature extraction?” Accessed: Jan. 09, 2025. [Online]. Available: https://www.educative.io/answers/what-is-feature-extraction

[26] C. L. Chowdhary and D. P. Acharjya, “Segmentation and Feature Extraction in Medical Imaging: A Systematic Review,” *Procedia Comput Sci*, vol. 167, pp. 26–36, Jan. 2020, doi: 10.1016/J.PROCS.2020.03.179.

[27] S. Liang *et al.*, “Edge YOLO: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles,” *ieeexplore.ieee.org*, vol. PP, 2022, doi: 10.1109/TITS.2022.3158253.

[28] X. Zhao and S. Zhang, “A Review on Facial Expression Recognition: Feature Extraction and Classification,” *IETE Technical Review*, vol. 33, no. 5, pp. 505–517, 2016, doi: 10.1080/02564602.2015.1117403.

[29] U. R. Aparna and S. Paul, “Feature selection and extraction in data mining,” *Proceedings of 2016 Online International Conference on Green Engineering and Technologies, IC-GET 2016*, May 2017, doi: 10.1109/GET.2016.7916845.

[30] E. G.-E. V. Alliance and undefined 2012, “Introduction to computer vision using opencv,” *bdti.comE GregoriEmbedded Vision Alliance, 2012•bdti.com*, 2012, Accessed: Jan. 14, 2025. [Online]. Available: https://www.bdti.com/MyBDTI/pubs%3ABDTI_ESCSV_2012_Intro_Computer_Vision.pdf

[31] “Line detection — Basics of Image Processing.” Accessed: Jan. 14, 2025. [Online]. Available: https://vincmazet.github.io/bip/detection/lines.html

[32] Q. Zhao, Q. Peng, and Y. Zhuang, “Lane line detection based on the codec structure of the attention mechanism,” *J Real Time Image Process*, vol. 19, no. 4, pp. 715–726, Aug. 2022, doi: 10.1007/S11554-022-01217-Z/TABLES/7.

[33] R. Pautrat, D. Barath, V. Larsson, M. R. Oswald, and M. Pollefeys, “DeepLSD: Line Segment Detection and Refinement With Deep Image Gradients,” 2023. Accessed: Jan. 14, 2025. [Online]. Available: https://github.com/cvg/DeepLSD.

[34] L. Zhou, W. Y.-J. of C. N. and, and undefined 2022, “Improved Convolutional Neural Image Recognition Algorithm based on LeNet-5,” *Wiley Online LibraryL Zhou, W YuJournal of Computer Networks and Communications, 2022•Wiley Online Library*, vol. 2022, 2022, doi: 10.1155/2022/1636203.

[35] T. Ehret, J. M.-I. P. O. Line, and undefined 2024, “Line Segment Detection: a Review of the 2022 State of the Art,” *ipol.imT Ehret, JM MorelImage Processing On Line, 2024•ipol.im*, vol. 14, pp. 41–63, 2024, doi: 10.5201/ipol.2024.481.

[36] P. Bharti, A. Ayush Kumar, R. Scholar, and A. Professor, “Study of various Approaches used for Object Detection and Recognition by AI and Natural Language Processing,” *ijtrm.comP Bharti, A Kumarijtrm.com*, vol. 11, pp. 2348–9006, 2024, Accessed: Jan. 08, 2025. [Online]. Available: https://www.ijtrm.com/PublishedPaper/11Vol/Issue8/2024IJTRM8202445403-94062000-851e-4352-b3e5-ff0f3fe25aa35002.pdf

[37] “Object Detection vs. Object Recognition: What’s the Difference?” Accessed: Jan. 08, 2025. [Online]. Available: https://www.augmentedstartups.com/blog/object-detection-vs-object-recognition-what-s-the-difference?srsltid=AfmBOoq-GkD5ye0XyUyrCp4DeNUNFgnY6fjgb0VgV0NKTbxjI3rrEUzM

[38] Y. Amit, P. Felzenszwalb, R. G.-C. V. A. R. Guide, and undefined 2021, “Object detection,” *Springer*, Accessed: Jan. 08, 2025. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-030-63416-2_660.pdf

[39] M. Petroniu, … M. T.-2015 20th I., and undefined 2015, “Object recognition with kinect sensor,” *ieeexplore.ieee.org*, Accessed: Jan. 08, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7168458/

[40] S. Abba, A. Bizi, J. Lee, S. Bakouri, M. C.- Heliyon, and undefined 2024, “Real-time object detection, tracking, and monitoring framework for security surveillance systems,” *cell.com*, Accessed: Jan. 08, 2025. [Online]. Available: https://www.cell.com/heliyon/fulltext/S2405-8440(24)10953-X

[41] R. Lys, Y. O.-A. in C.-P. Systems, and undefined 2023, “Development of a Video Surveillance System for Motion Detection and Object Recognition,” *science.lpnu.ua*, vol. 8, no. 1, p. 2023, doi: 10.23939/acps2023.01.050.

[42] S. Rani, D. Ghai, S. Kumar, M. P. Kantipudi, A. H. Alharbi, and M. A. Ullah, “Efficient 3D AlexNet architecture for object recognition using syntactic patterns from medical images,” *Wiley Online Library*, vol. 2022, 2022, doi: 10.1155/2022/7882924.

[43] C. Wang, B. Zhu, C. Huang, and L. Zhao, “Real-Time Efficient Retail Object Recognition,” *2023 International Conference on Platform Technology and Service, PlatCon 2023 - Proceedings*, pp. 30–35, 2023, doi: 10.1109/PLATCON60102.2023.10255182.

[44] E. Maiettini, G. Pasquale, L. Rosasco, and L. Natale, “On-line object detection: a robotics challenge,” *Auton Robots*, vol. 44, no. 5, pp. 739–757, May 2020, doi: 10.1007/S10514-019-09894-9/FIGURES/10.

[45] “Object Detection 101: Applications, Challenges, and Future Directions · Neil Sahota.” Accessed: Jan. 08, 2025. [Online]. Available: https://www.neilsahota.com/object-detection-101-applications-challenges-and-future-directions/?utm_source=chatgpt.com

[46] “AI Object Detection in 2025: Definitive Guide - Daten & Wissen.” Accessed: Jan. 08, 2025. [Online]. Available: https://datenwissen.com/blog/ai-object-detection/

[47] V. Badrinarayanan, … A. K.-I. transactions on, and undefined 2017, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *ieeexplore.ieee.orgV Badrinarayanan, A Kendall, R CipollaIEEE transactions on pattern analysis and machine intelligence, 2017•ieeexplore.ieee.org*, Accessed: Jan. 04, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7803544/

[48] J. Long, E. Shelhamer, T. D.-P. of the IEEE, and undefined 2015, “Fully convolutional networks for semantic segmentation,” *openaccess.thecvf.com*, Accessed: Jan. 04, 2025. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html

[49] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.

[50] L.-C. Chen, G. Papandreou, S. Member, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *ieeexplore.ieee.orgLC Chen, G Papandreou, I Kokkinos, K Murphy, AL YuilleIEEE transactions on pattern analysis and machine intelligence, 2017•ieeexplore.ieee.org*, Accessed: Jan. 04, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7913730/

[51] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, “SegFormer: Simple and efficient design for semantic segmentation with transformers,” *proceedings.neurips.ccE Xie, W Wang, Z Yu, A Anandkumar, JM Alvarez, P LuoAdvances in neural information processing systems,*

2021•proceedings.neurips.cc, Accessed: Jan. 04, 2025. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/64f1f27bf1b4ec22924fd0acb550c235-Abstract.html

[52] "Instance Segmentation in Computer Vision [2024 Overview] | Encord." Accessed: Jan. 04, 2025. [Online]. Available: https://encord.com/blog/instance-segmentation-guide-computer-vision/

[53] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *Int J Multimed Inf Retr*, vol. 9, no. 3, pp. 171–189, Sep. 2020, doi: 10.1007/S13735-020-00195-X.

[54] K. He, G. Gkioxari, … P. D.-P. of the I., and undefined 2017, "Mask r-cnn," *openaccess.thecvf.com*, Accessed: Jan. 04, 2025. [Online]. Available: http://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html

[55] D. Bolya, C. Zhou, F. Xiao, Y. L.-P. of the IEEE, and undefined 2019, "Yolact: Real-time instance segmentation," *openaccess.thecvf.com*, Accessed: Jan. 04, 2025. [Online]. Available: http://openaccess.thecvf.com/content_ICCV_2019/html/Bolya_YOLACT_Real-Time_Instance_Segmentation_ICCV_2019_paper.html

[56] Z. Tian, C. Shen, H. C.-C. V. 2020: 16th European, and undefined 2020, "Conditional convolutions for instance segmentation," *SpringerZ Tian, C Shen, H ChenComputer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020•Springer*, Accessed: Jan. 04, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-58452-8_17

[57] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," *openaccess.thecvf.com*, Accessed: Jan. 04, 2025. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Kirillov_Panoptic_Segmentation_CVPR_2019_paper.html

[58] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," *proceedings.neurips.ccE Xie, W Wang, Z Yu, A Anandkumar, JM Alvarez, P LuoAdvances in neural information processing systems, 2021•proceedings.neurips.cc*, Accessed: Jan. 14, 2025. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/64f1f27bf1b4ec22924fd0acb550c235-Abstract.html

[59] L.-C. Chen, G. Papandreou, S. Member, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *ieeexplore.ieee.orgLC Chen, G Papandreou, I Kokkinos, K Murphy, AL YuilleIEEE transactions on pattern analysis and machine*

intelligence, 2017•ieeexplore.ieee.org, Accessed: Jan. 14, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7913730/

[60] C. Zhang *et al.*, "A Comprehensive Survey on Segment Anything Model for Vision and Beyond," May 2023, Accessed: Jan. 14, 2025. [Online]. Available: https://arxiv.org/abs/2305.08196v2

[61] A. Kirillov *et al.*, "Segment Anything," 2023. Accessed: Jan. 17, 2025. [Online]. Available: https://segment-anything.com.

[62] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast Semantic Segmentation Network," *30th British Machine Vision Conference 2019, BMVC 2019*, Feb. 2019, Accessed: Jan. 14, 2025. [Online]. Available: https://arxiv.org/abs/1902.04502v1

[63] H. Shu *et al.*, "TinySAM: Pushing the Envelope for Efficient Segment Anything Model," Dec. 2023, Accessed: Jan. 17, 2025. [Online]. Available: http://arxiv.org/abs/2312.13789

[64] "GitHub - Praneet9/SegFormer_Segmentation: Semantic Segmentation with SegFormer on Drone Dataset." Accessed: Sep. 03, 2025. [Online]. Available: https://github.com/Praneet9/SegFormer_Segmentation

9 Annexe

9.1 Literature Review Methodology

This section outlines the systematic approach adopted for the literature review, detailing the central research questions, the inclusion and exclusion criteria for selecting studies, and the strategy employed to ensure the incorporation of the most relevant and up-to-date sources. Establishing clear guidelines ensures the construction of a comprehensive and reliable knowledge base, providing a robust foundation for the research.

The literature review shows four main research themes. The first is advanced computer vision, where recent deep learning models such as Transformers and attention mechanisms are explored to enhance accuracy and efficiency in detection tasks. The second is segmentation, a prevailing theme for extracting precise boundaries and supporting detailed pixel-level analysis for sail line detection. Third, a theme involves the difficulties of outdoor and maritime environments that address robustness to varying illumination, motion, and weather conditions. Lastly, the literature includes studies on object recognition and feature extraction for generic objects that, although not necessarily domain-specific, provide essential frameworks and techniques that can be transferred for sail shape detection.

9.1.1 Research Questions

The literature review was structured to address the following **core research questions**, each closely aligned with the **objectives of this thesis**:

1. **Which aerodynamic and structural aspects of sails are essential for understanding and measuring parameters such as camber, draft, and twist?**

   To explore this question, the review focused on documented studies discussing sail geometry and performance and the role of specific markers — such as orange sail lines placed on the sail — in quantifying aerodynamic parameters.

2. **What are the predominant computer vision methods for accurately detecting thin or narrow features in high-resolution images?**

   This question guided an examination of AI-based techniques, including line detection, feature extraction, object detection, and segmentation methodologies, emphasising their ability to process large-resolution frames effectively.

3. **How do state-of-the-art semantic segmentation algorithms compare in terms of accuracy, computational efficiency, suitability for real-time or near real-time deployment and capability of handling large-resolution frames?**

   Given that the system is designed for a tablet with a camera setup, this question focused on evaluating the trade-offs between model complexity and responsiveness, ensuring the selection of an approach capable of efficient onboard processing.

These research questions played a key role in shaping the scope and depth of the literature review, ensuring that the selection of studies directly addressed the technical and operational constraints of the thesis.

9.1.2 Inclusion and Exclusion Criteria

Inclusion and exclusion criteria were established to ensure the reviewed literature's relevance, reliability, and recency. This filtering process facilitated the selection of studies that provide the most up-to-date and pertinent information for the research.

Inclusion Criteria:

- Articles published in 2011 or later, ensuring coverage of recent advancements in CV and any new insights in the sailing and maritime domain.
- Reliable online resources from recognised institutions, professional associations, or established research groups, discussing sailing concepts or developments in CV.
- Peer-reviewed materials, including conference papers, journal articles, book chapters, or open-access platforms with robust editorial oversight.
- Studies providing tangible insights into topics such as line/feature detection, object detection & recognition, and segmentation models applicable to the thesis objectives.

Exclusion Criteria:

The following sources were excluded to maintain the relevance and applicability of the review:

- Publications dated before 2011, as older studies may not reflect current state-of-the-art techniques in CV and machine learning.
- Materials unrelated to sailing, CV, or advanced image processing ensure the review focuses on thematically relevant sources.

This filtering process helped develop a focused dataset of studies offering the most up-to-date and contextually pertinent information in CV and maritime performance analysis.

9.1.3 Search Strategy

A systematic search approach was employed across various digital libraries, preprint repositories, and trusted online platforms to ensure comprehensive coverage and relevance. This approach facilitated the identification of **high-quality sources** aligned with the research objectives. Table 8 presents the primary resources used, while Table 9 outlines the core topics and keywords leveraged in the search process.

Table 8 - Resources Used in the Search Process.

Resource Name	Type
IEEE Xplore	Digital Library/Repository
SpringerLink	Digital Library
MDPI Open-Access	Publisher
ArXiv Preprint	Repository
Google Scholar	Search Engine/Academic Index
Websites	Official Sourcing (e.g., sailing orgs)

Table 9 - Topics and Relevant Words.

Topic	Relevant Words
Sailing/Maritime Context	"sail geometry", "sail shape", "sail trimming", "sail performance", "aerodynamic properties in sailing", "aerodynamic features in sailing", "sailing dynamics", "draft", "twist", "camber", "wind conditions", "boat performance", "marine environment", "maritime environment", "marine line detection", “wind conditions”
Computer Vision	“thin line detection,” “line segment detection,” “gradient-based detection,” “lane detection,” “image gradients,” “refinement of line segments,” “feature extraction,” “object detection vs object recognition,” “object detection,” “object recognition,” “image processing fundamentals,” “boat detection,” “pose estimation,” “OpenCV basics,” “machine learning in computer vision,” “classical CV algorithms,” “pattern matching”, "computer vision fundamentals"
Advanced Image Processing Techniques	“semantic segmentation,” “instance segmentation,” “panoptic segmentation,” “transformer-based segmentation,” “deep convolutional encoder-decoder,” “real-time image segmentation,” “FCN,” “attention-based segmentation,” “pixel-level labelling,” “high-resolution computer vision,” “global vs. local context,” “resource-efficient segmentation,” “attention mechanism for large images,” “edge cloud cooperation,” “multi-scale processing,” “real-time HD video processing,” “real-time HD video analysis,” “image tiling”, "conditional convolutions"

AI-based algorithms	“low-latency inference,” “optimised CNN,” “Fast-SCNN,” “lightweight architecture,” “on-device computer vision,” “embedded GPU,” “edge AI,” “DeepLab,” “U-Net,” “computational cost,” “near real-time detection,” “vision transformer,” “SegFormer,” “Segment Anything Model,” “transformer-based segmentation,” “transformer pipeline,” “attention mechanisms,” “self-attention,” “global context in segmentation” “accuracy” “efficient” “handling large-images”, "accuracy vs efficiency", "DeepLSD line detection", “SAM”

Methods to Enhance Search Results

1. **Cross-referencing citations**: Relevant articles were identified by examining the bibliographies of the most relevant studies. Both backwards and forward citation tracking were utilised to locate additional research contributions aligned with the thesis objectives.

2. **Website verification**: Reliable sailing or CV websites were scanned for practical and real-world examples of image processing solutions in maritime contexts.

9.1.4 Search Results

A total of 64 references were identified as directly relevant to the scope of this review. These sources reach a range of topics, including sailing dynamics, CV methodologies, and the application of advanced algorithms for large-scale or real-time image processing.

A breakdown of the selected references is as follows:

1. **Publication Date Analysis**

    - A significant **30%** of the references were published in **2024**, highlighting growing research interest in advanced segmentation models and their applications in maritime environments. These studies contribute cutting-edge insights into deep learning advancements and image processing techniques.
    - Around **40%** appeared between **2020 and 2023**, reflecting the rapid evolution of deep learning, attention-based architectures, and specialised applications in embedded CV.
    - The remaining **30%** predate 2020 but stay within the 2011–2019 range. These works provide foundational concepts and seminal approaches to image segmentation.

2. **Primary Publishers/Platforms**

    - **IEEE Xplore** contributed roughly **50%** of the reviewed peer-reviewed articles, showcasing consistent coverage of state-of-the-art CV and image processing.
    - **SpringerLink** and **MDPI** collectively account for about **20%**, delivering scholarly journals and open-access research with a strong focus on engineering and applied sciences.
    - **ArXiv** provides preprints for approximately **10%**.
    - The remaining includes specialised websites and direct references from recognised institutions or professional organisations in sailing.

3. **Research Themes**

    - **Advanced Computer Vision**: Roughly **35%** of the literature discusses cutting-edge deep learning or AI-driven detection/segmentation, covering the latest architectures (e.g., Transformers, attention mechanisms) and optimisation techniques for large-scale inference.
    - **Segmentation**: About **25%** focuses specifically on refined modern segmentation approaches.
    - **Maritime/Outdoor Environments**: Close to **20%** addresses real-world deployment challenges, including high-resolution video, moving platforms, or harsh weather conditions.
    - **General Object Recognition and Feature Extraction**: The remaining **20%** covers feature-based methods, object detection strategies, and foundational CV frameworks that can be adapted to various contexts.

4. **Geographical Distribution**

    - A significant number of the studies (approximately **40%**) originate from authors or research groups in **the Asia-Pacific region**, reflecting the region's robust engagement with emerging AI technologies.
    - **North America** (about **30%**) and **Europe** (roughly **25%**) also represent major contributors, with a focus on practical AI applications and theoretical advancements in deep learning.
    - The remaining **5%** stems from other locales or multinational collaborations, indicating global interest in these topics.

These aggregated findings represent a diverse yet thematically cohesive body of research, providing a robust foundation for understanding sailing-specific and CV-based perspectives. This thesis's subsequent sections will leverage established methodologies and recent innovations, ensuring a versatile approach to the research objectives. It is important to note that the literature review remains an ongoing process. As the thesis progresses, this section will continuously expand and refine, incorporating newly published research and relevant advancements to maintain the most up-to-date and comprehensive perspective.