Journal of
Manufacturing and Materials Processing

Article

Real-Time Stringing Detection for Additive Manufacturing

Oumaima Charia [1,*], Hayat Rajani [2], Inés Ferrer Real [3], Miquel Domingo-Espin [1] and Nuno Gracias [2]

[1] Eurecat, Centro Tecnológico de Cataluña, Advanced Manufacturing Systems Unit, Av. Universitat Autònoma, 23, 08290 Cerdanyola del Vallès, Catalonia, Spain; miquel.domingo@eurecat.org
[2] Computer Vision and Robotics Research Institute (VICOROB), University of Girona, Campus Montilivi, Edifici P4, 17003 Girona, Catalonia, Spain; hayat.rajani@udg.edu (H.R.); ngracias@silver.udg.edu (N.G.)
[3] Department of Mechanical Engineering and Industrial Construction, University of Girona, Campus Montilivi, Edifici P1, 17003 Girona, Catalonia, Spain; ines.iferrer@udg.edu
* Correspondence: oumaima.charia@eurecat.org

Abstract: Additive Manufacturing (AM), commonly known as 3D printing, has gained significant traction across various industries due to its versatility and customization potential. However, the process remains time-consuming, with print durations ranging from hours to days depending on the complexity and size of the object. In many cases, errors occur due to object misalignment, material stringing due to nozzle overflow, and filament blockages, which can lead to complete print failures. Such errors often go undetected for extended periods, resulting in substantial losses of time and material. This study explores the implementation of traditional computer vision, image processing, and machine learning techniques to enable real-time error detection, specifically focusing on stringing-related anomalies. To address data scarcity in training machine learning models, we also release a new dataset and improve upon the results achieved by the Obico server model, one of the most prominent tools for stringing detection. Our contributions aim to enhance process reliability, reduce material wastage, and optimize time efficiency in AM workflows.

Keywords: additive manufacturing; error detection; machine learning; computer vision; image processing; blender

check for updates

Academic Editor: David L. Sales

Received: 28 November 2024
Revised: 14 February 2025
Accepted: 17 February 2025
Published: 25 February 2025

Citation: Charia, O.; Rajani, H.; Ferrer Real, I.; Domingo-Espin, M.; Gracias, N. Real-Time Stringing Detection for Additive Manufacturing. *J. Manuf. Mater. Process.* **2025**, *9*, 74. https://doi.org/10.3390/jmmp9030074

1. Introduction

Additive Manufacturing (AM) [1], commonly known as 3D printing, broadens manufacturing possibilities by allowing the creation of intricate geometrical shapes through design software. This technology has driven innovation across a wide range of industries including biomedical, construction and aerospace, and many others.

The 3D printing process involves three key steps: designing the model, slicing, and printing. Unlike traditional manufacturing methods, designing for 3D printing starts with creating a digital 3D model using Computer-aided Design (CAD) software, where designers must consider the unique strengths and constraints of AM, such as layer adhesion, support structures, and material properties. This step allows for significant design flexibility, enabling the creation of complex geometries, internal structures, multi-material designs, part consolidation, and mass customization that are often impossible with traditional techniques like injection molding or CNC machining.

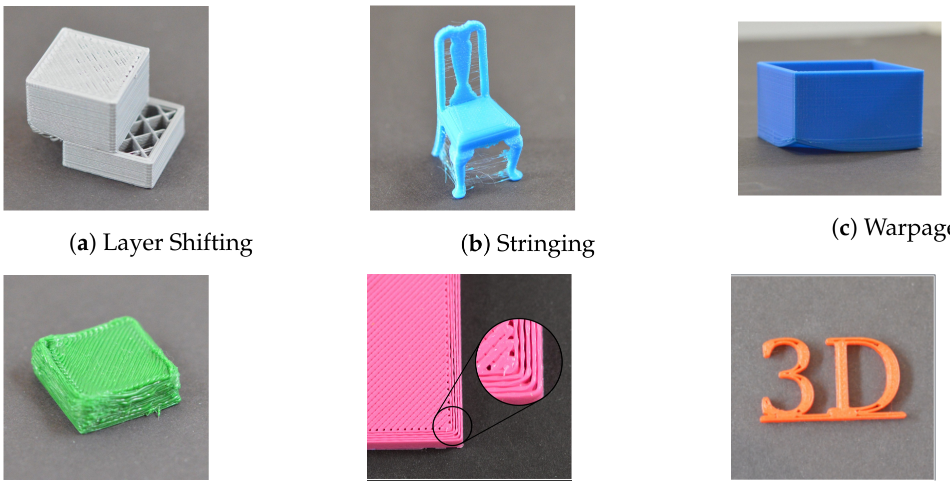
Among the various AM technologies, this paper focuses specifically on Fused Filament Fabrication (FFF), which has become one of the most popular 3D printing techniques due to its accessibility, material versatility, and low-cost equipment. FFF works by extruding thermoplastic material through a heated nozzle, which deposits it layer by layer onto a build platform. During the process, a polymer filament is melted in the nozzle at a temperature slightly above its melting point, then deposited onto the printer's hotbed under computer control, fusing it with the adjacent layers below. To prepare a model for FFF printing, the design is imported into slicing software, which divides it into horizontal layers, and generates toolpaths. At this stage users can adjust various manufacturing parameters, including layer height, print speed, infill density, and support structures to optimize both the print quality and production efficiency. The resulting toolpaths, usually saved as a file in G-code format, is then transferred to the 3D printer. The printer reads the G-code instructions to move the print head and extrude material layer by layer, gradually building the physical object. Throughout the printing process, the machine follows these instructions to ensure an accurate reproduction of the digital model.

In FFF, as the nozzle moves along the X,Y and Z axes, the semi-liquid filament is precisely deposited to form each layer of the object. To ensure strong layer bonding, the heated bed maintains the material at an optimal temperature, preventing premature cooling. FFF commonly uses materials such as Acrylonitrile Butadiene Styrene (ABS), Polylactic Acid (PLA), Polyamide (PA), Polycarbonate (PC), Polyurethane (PU) and High-density Polyethylene (HDPE), with ABS and PLA being the most popular due to their affordability, availability, and ease of use. FFF is well-suited for fabricating small-batch parts where high accuracy and surface finish are critical.

FFF printing is susceptible to various errors, as depicted in Figure 1, due to its reliance on precise control of the deposition and movement of the material. Warping [2], for instance, occurs when parts cool unevenly, causing edges or corners to lift from the build platform, often due to improper bed temperature or inadequate adhesion of the first layer. Another frequent issue is stringing [3], where excess filament creates thin strands between different sections of the print as the nozzle moves. This can usually be mitigated by adjusting the retraction settings. Layer shifting, is often due to loose belts, stepper motor issues, high print speeds, bed obstructions, or software errors, causing unintended deviations in the print layers. Under-extrusion [4] is another common problem where insufficient material is extruded, leading to weak layer bonding or gaps in the print. This is often caused by clogged nozzles or incorrect filament settings. Conversely, over-extrusion [5] can deposit too much material, causing blobs, rough surfaces, and dimensional inaccuracies. Lastly, Gaps may arise from factors such as high layer height or improper infill settings, resulting in visible layer lines or rough surfaces.

Despite the range of defects in the FFF process, this study focuses specifically on the detection of *stringing*. As one of the most prevalent issues in FFF. Stringing not only degrades surface finish and dimensional accuracy but also results in material waste and increase post-processing time due to the labor-intensive task of manual removal. These challenges make addressing stringing essential for producing high-quality 3D printed objects across various materials and print settings.

To counteract stringing and other defects, the classical approach involves a skilled worker who must monitor the AM process, recognize an error, stop the printing, remove the part, and then appropriately adjust the parameters for a new part. This process is time-consuming, especially with new materials or printers, and errors may still be missed if the worker is not continuously observing each process. This has motivated diverse and innovative research to develop notable approaches for detecting errors in real-time, mitigating these challenges and optimizing overall efficiency, which will now be described.



(**a**) Layer Shifting (**b**) Stringing (**c**) Warpage

(**d**) Over Extrusion (**e**) Under Extrusion (**f**) Gaps

Figure 1. Common types of issues in Additive Manufacturing. Figure adapted from [6].

Several studies have focused on machine learning techniques to enhance defect detection. Delli and Chang [7] propose a method based on Support Vector Machines (SVMs) to automatically assess the quality of 3D printed parts using Fused Deposition Modeling (FDM). At predefined checkpoints based on the geometry of the part being printed, images are collected, preprocessed and passed to an SVM model, which classifies the part as *good* or *bad*. However, this requires the printing process to be temporarily paused while the print bed and nozzle are repositioned to capture the image. Yean and Chew [8], on the other hand, employ an AlexNet-SVM model to detect stringing and spaghetti defects in FDM 3D printing. The model was trained on a three-class dataset (stringing, spaghetti, no defect) and deployed on a Raspberry Pi 3 for real-time classification. However, the authors report a drop in performance for real-life scenarios, and recommend further optimization and fine-tuning of the model. Karna et al. [9], in contrast, approach the problem as a detection rather than a classification task. They present an enhanced YOLOv8 [10] model with additional feature extraction layers for improved fault detection in FDM-based 3D printers. Their approach carefully balances model complexity with performance gains, demonstrating its potential to improve fault detection in smart additive manufacturing environments. Similarly, Paraskevoudis et al. [11] approach the problem as a detection task and employ a Single Shot Detector, specifically for the detection of stringing in FFF 3D printing. Although the model demonstrated reasonable performance on images similar to the training set, it was unable to generalize well to external datasets. Likewise, Obico [12] has developed a detection model specifically tailored for detecting spaghetti defects in 3D printing. They employ a YOLOv2 [13], trained via the Darknet framework [14], with open code and model weights. In addition, they offer a platform that enables remote monitoring and control of the printing process in real-time.

In parallel, other researchers have emphasized the application of computer vision techniques for real-time quality assessment in FDM printing. For instance, AbouelNour and Gupta [15] integrates in-situ multi-sensor monitoring using both optical imaging and infrared thermography to detect defects during the additive manufacturing process. By modifying the G-code to embed defects and enable time lapse imaging, they used an off-axis multi sensor system that combines optical and infrared thermography. This system captures images and detects temperature variations, where optical image processing identifies surface defects, and thermal imaging revealed internal defects caused by voids.

Charalampous et al. [16] introduced a vision-based method to evaluate additive manufacturing by comparing the printed part with its digital 3D model in real-time. The G-code is converted into a reference 3D point cloud. During printing, 3D scans generate point clouds of the product, which are filtered, segmented, and registered. These scanned point clouds are compared with the reference model, and if dimensional deviations exceed a set threshold, the process is automatically halted. Additionally, Petsiuk and Pearce [17] presented an open source software algorithm and hardware structure for tracking and correcting 3D printing errors in real-time. Their methodology used three main branches of image processing: side view height validation to detect issues such as "blocked nozzle", "lack of material", and "major deformations"; Global trajectory correction using Multi-Template Matching (MTM) [18] to track significant horizontal and vertical displacements; and Iterative Closest Point (ICP) algorithm [19] to adjust displacements and local texture analysis to identify anomalies through clustering techniques. This approach enables real-time monitoring and corrective actions during the printing process. Similarly, Baumann and Roller [20] captured video during the 3D printing process and selected three consecutive frames. They applied segmentation by colour and computed the differential between the frames to check the horizontal changes. This approach is used to detect two types of failures: "Detachment" & "Deformation of the object". For "missing material flow" error, a bounding box is placed on the object in the binary image, and the algorithm checks the deviation in detecting the center of the bounding box. Petsiuk and Pearce [21] developed an open-source method for detecting 3D printing anomalies using a monocular camera to capture real-time data. They generated ideal synthetic reference images in Blender and compared them with real images using top-down views. The real image is divided into small square cells, each converted into a nine-channel Histograms of Histograms of Oriented Gradients (HOG) [22] feature vector. These vectors are combined into 2×2 blocks for robustness, and the same process is applied to the reference image. A similarity map is then generated, assigning numerical values to each section of the real image based on its proximity to the ideal print. These values are colour-coded to highlight areas with potential defects.

The aim of our approach is to develop a robust and generalizable framework for detecting *stringing* defects across diverse and unpredictable scenarios by independently leveraging both machine learning and computer vision techniques in order to overcome the limitations of previous methods. For the machine learning approach, similar to Paraskevoudis et al. [11], we frame the problem as a detection task and focus solely on stringing defects. However, unlike Yean and Chew [8], we combine spaghetti and stringing defects into a single class due to their visual similarities, and in order to avoid suboptimal performance caused by class imbalance. We also generated a much larger training dataset comprising objects of diverse shapes, structures and colours to improve the model's generalizability across different real-world scenarios. Additionally, rather than training the model from scratch, we leveraged the pre-trained model by Obico [12], fine-tuning it to improve both detection accuracy and generalization across various types of stringing defects. This not only resulted in achieving better robustness but also significantly reduced the training time. For the computer vision approach, similarly to the method of Petsiuk and Pearce [21], we compared real-time images with synthetic ones to highlight any discrepancies. The synthetic images are rendered from G-code in Blender but, instead of using complex shaders, we focused on binary masks to simplify the rendering process and reduce the computational overhead. Furthermore, in contrast to the method of Petsiuk and Pearce which requires a specialized printing and illumination setup, our method is tailored to conventional 3D printers with no particular illumination requirement. As such, our method can be used in a wide range of existing printing platforms.

This paper advances the state-of-the-art in stringing detection for additive manufacturing through two key contributions. First, it introduces and evaluates a novel approach that integrates two complementary methods for stringing detection. The first method uses a machine learning model with transfer learning applied to an existing pre-trained network, enabling efficient feature extraction and classification. The second method employs standard computer vision techniques, to compute silhouette differences and detect deviations from the expected printing plan. Notably, these methods operate without requiring specialized printing setups, relying only on one or more standard cameras. Second, the paper addresses a critical issue that is currently preventing wider use of machine learning in AM error detection. This issue is the lack of well annotated training datasets. This gap is addressed in this paper, as it provides a publicly available dataset that is the first of its kind, enabling further research and development in this area.

The remainder of the paper is organized as follows. Section 2 presents an overview of the datasets generated for this work. Next, Section 3 provides details on the adopted methodology and the experimental setup. This is followed by the presentation and visualization of results in Section 4. Finally, Section 5 concludes this study and outlines the directions for future research.

2. Datasets

This section describes the datasets utilized to train and evaluate the machine learning model and the computer vision approach. We used a dual camera setup, where a microscope-type camera was positioned in front of the 3D printer, while a conventional industrial camera (referred to in this paper as FLIR) was mounted at a 90-degree angle to the microscope on the side of the printer. This setup allowed us to capture the printout from multiple perspectives, ensuring that stringing defects could be observed from various angles. The same camera mounting setup was employed for both approaches.

2.1. Machine Learning

After each layer is printed, the printer sends a command to the host computer, initiating time-lapse photography. This triggers both cameras to capture images and save them to the disk. It also causes the print head to move to the side and the print bed to move to the front. As the print head moves, the filament continues to ooze, deliberately inducing stringing.

A total of 28 sequences were printed, each captured by both cameras, with a diverse set of objects in various colours and shapes, resulting in a dataset of 5342 images (1280 $\times$ 720 resolution) capturing layers with stringing errors during the printing process. This diversity in object shapes and colours ensures that the datasets are representative of real world 3D printing scenarios, providing a robust basis for training and evaluating the model. From the 56 collected sequences, 46 sequences were selected to form the training set, comprising 4217 images. These images were manually annotated using Roboflow [23] for its ease of labeling and ability to export data in various desired formats. The remaining 10 sequences, comprising 1125 images were used as the validation set. Figure 2 depicts some examples of the annotated images.

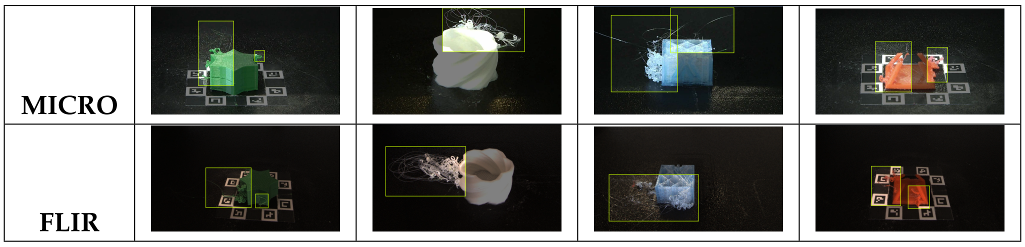


Figure 2. Examples of annotated images used for training the ML model, captured by the microscope (**top**) and a FLIR camera (**Bottom**).

Additionally, we collected 15 more sequences without explicitly inducing stringing. The resulting error-free images were used as negative samples during training. Of these, 12 sequences were incorporated in the training set, and 3 sequences were included in the validation set. This resulted in a total of 5515 training images and 1419 validation images.

The datasets are publicly available on Zenodo (accessed on 21 January 2025) (https://doi.org/10.5281/zenodo.14711320) and are organized as depicted in Figure 3. The root data directory consists of two subdirectories, called *sequences* and *yolo*. The former contains all the individual sequences that were collected, further organized into subdirectories based on whether they have stringing or not. Each of these subdirectories contains images captured by the camera and the microscope. The latter comprises the training and validation subsets for the detection model, along with YOLO-style annotations.

```
datasets
├── sequences
│   ├── with_stringing
│   │   ├── sequence_1
│   │   │   ├── flir/
│   │   │   └── micro/
│   │   ├── sequence_2
│   │   │   ├── flir/
│   │   │   └── micro/
│   │   ├── ⋮
│   │   └── sequence_28
│   │       ├── flir/
│   │       └── micro/
│   └── without_stringing
│       ├── sequence_3
│       │   ├── flir/
│       │   └── micro/
│       ├── sequence_5
│       │   ├── flir/
│       │   └── micro/
│       ├── ⋮
│       └── sequence_28
│           ├── flir/
│           └── micro/
└── yolo
    ├── train/
    └── valid/
```

Figure 3. Structure of the data repository.

2.2. Computer Vision

For the computer vision task, binary images were required for the dataset. These were generated using Blender's emission shader. The object was selected, and an emission shader was applied with the emission colour set to 255, connected to the surface input of the material output node. For the background, the emission shader was set to 0 and linked to the surface input of the material output node. This process yielded the necessary binary images for the dataset. To evaluate the computer vision approach, six sequences from machine learning datasets were used.

3. Methodology

3.1. Machine Learning Approach

We approach the problem as a detection task rather than image classification, specifically for identifying stringing defects in frames of video sequences collected during the printing process. We fine-tune the YOLOv2 model from Obico [12] on our generated datasets. In order to main compatibility with the original implementation, we opted to retain the Darknet framework [14], preserving the model weights in their native format and avoiding the need for conversion to modern frameworks. The model was fine-tuned using Google Colab [24] on a Tesla T4 GPU for 320 iterations with a batch size of 64. We employed the Adam optimizer with a weight decay of 5×10^{-4} and a learning rate of 5×10^{-4}, decayed using a Stochastic Gradient Descent with Restarts (SGDR) scheduler following a cycle of 1600 iterations. Images were resized to 640×352 pixels and standard data augmentation techniques such as random rotation, random resized crop, and random horizontal and vertical flip were adopted. Throughout the training process, a gradual decrease in loss was observed until convergence, indicating effective learning with the chosen hyperparameters. The source code with all hyperparameter configurations and pre-trained models will be made available at https://github.com/CIRS-Girona/stringing-detection (accessed on 15 February 2024).

In addition, to compare performance, we trained YOLOv2 [13] and YOLOv4 [10] models from scratch, and a YOLOv4 model fine-tuned on COCO weights. Section 4 presents a comparison of these results.

All trained models were evaluated on a standard laptop equipped with an Intel Core i7-13850HX CPU operating at 2.10 GHz running Windows 10. Model performance is reported in terms of mean average precision calculated at an IoU threshold of 0.5 (mAP50), as well as precision, recall and F1-score. To guage the real-time performance of the models, we also report the inference speed in terms of number of frames processed by the models per second.

3.2. Computer Vision Approach

The computer vision solution included the development of a prototype using two cameras: One monitoring the front of the printer and the other monitoring the side. This setup allows for capturing images after each layer was printed, and to use those images for comparison with a reference image so as to highlight any misalignment. In order to carry out this image comparison, it was necessary to calibrate the cameras, perform image segmentation, and analyze the difference against the reference image. The flowchart below Figure 4 summarizes the steps that were followed in this part of our project. The process is explained in detail in the following subsections.

3.2.1. Scene Setup

For the real scene, eight ArUco markers [25] were created from the 4×4 dictionary with squares of size 4 mm. These markers are 2D binary-encoded fiducial patterns designed to be quickly located by computer vision systems. The printer bed was used as the reference frame. The markers were placed at the center of the printer bed, and their coordinates were recorded relative to the bed's origin.

To align with Blender's coordinate system, the unit system in Blender was switched to millimeters, and a planar mesh matching the printer bed's dimensions was created. The markers were then manually placed at the same coordinates as in the pysical setup. An add-on [26] was installed to allow importing the G-code. Camera calibration parameters were imported to position the virtual cameras accurately, ensuring their placement matched the real-world setup relative to the markers. This process ensured precise alignment between the real-world coordinate system and the virtual environment, enabling accurate representation and analysis (see Figure 5).

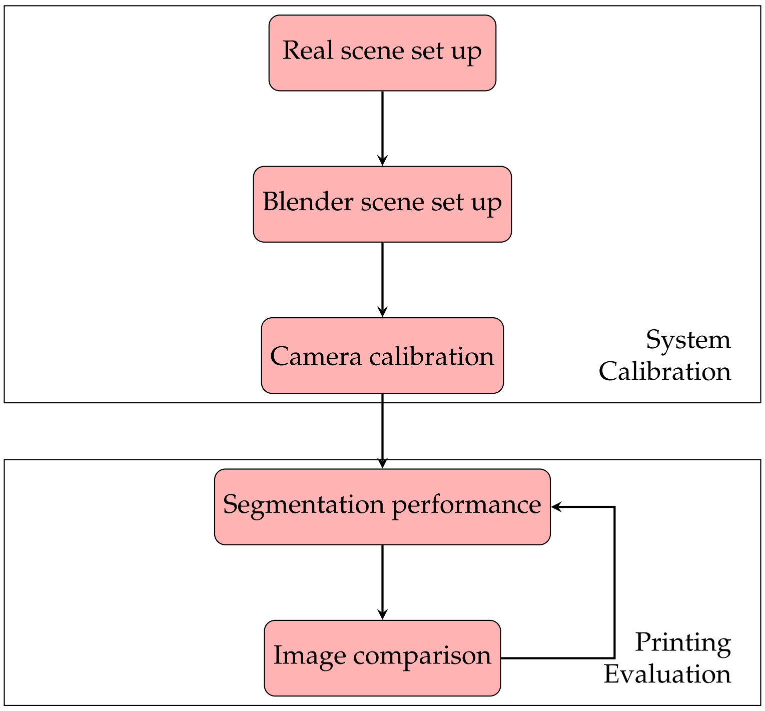


Figure 4. Procedure to fault detection for AM using computer vision.

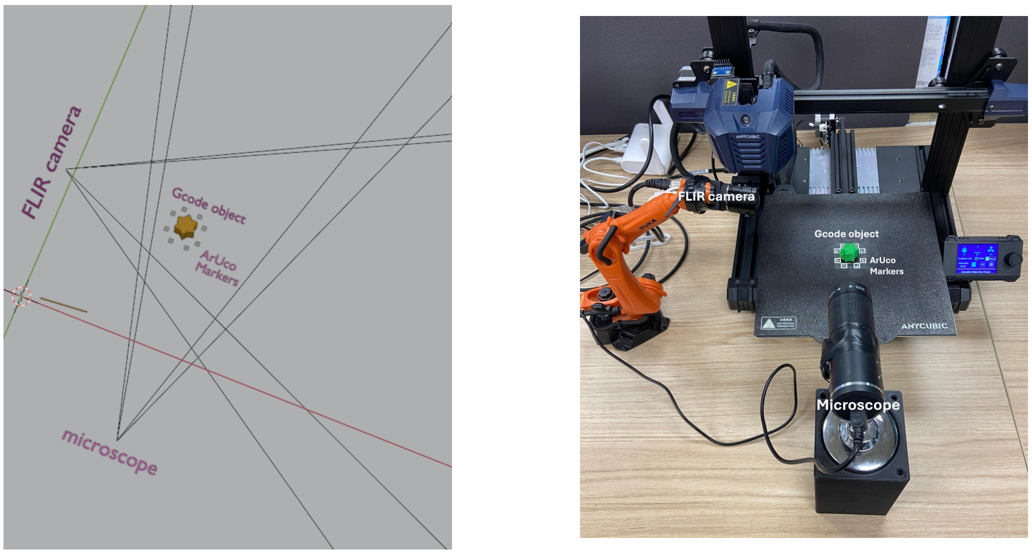


Figure 5. Blender scene (**Left**), real scene (**Right**).

3.2.2. Camera Calibration

The process began by generating a 6 × 6 checkerboard with squares sized 4 × 4 mm. 20 images of the board were captured from different positions, and angles.

For each image, OpenCV functions were utilized to detect the inner corners defined by the intersections between black and white squares on the chessboard. Subsequently, we assigned world coordinates to each of these corners. Using these parameters, we used cv2.calibrateCamera() to estimate the intrinsic parameter which includes the matrix K (as denoted by Equation (1)) and the distortion coefficient vector.

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

where f_x and f_y are the focal lengths, c_x and c_y are the coordinates of the optical center, and s is the skew coefficient.

For obtaining the extrinsic parameters, the solvePnP method was applied. 3D coordinates of the markers were determined manually. OpenCV functions were used to detect the markers within the captured images and extract their corresponding 2D pixel coordinates. Having these data, the solvePnP function from OpenCV was applied. This method enabled the computation of rotation and translation vectors.

In order to calibrate the virtual cameras to match the real ones, an implemented function in BlenderProc [27] was used to import the intrinsic parameters into Blender. For the extrinsic parameters, the camera orientation was converted into Euler angles, as required by Blender. For the position, the values obtained from solvePnP were used directly.

3.2.3. Segmentation Performance

To isolate the object from the background in the real images, color segmentation was employed using the HSV color space. Despite controlling the background to be black, variations in illumination affected the appearance of colors, making RGB segmentation ineffective. The process involved converting the colored image into HSV format using OpenCV's cv2.cvtColor function and applying manual thresholding with the cv2.inRange function. The thresholding for Hue (H), Saturation (S), and Value (V) were dynamically adjusted using OpenCV trackbars, enabling interactive fine-tuning to accurately highlight the object of interest.

3.2.4. Image Comparison

Firstly, it was necessary to ensure alignment between the Blender output images and the real ones. To achieve this, a Python script was implemented which converted the binary real image to green and the binary Blender image to red. By overlaying these images, the aligned areas appeared yellow. Additionally, to highlight differences between the real image and the Blender image, morphological operations were used. Initially, the input was three images: Real binary image, Blender binary image and the original real image. A 15×15 kernel was defined for the morphological operations. For the Blender binary image, dilation and erosion were applied to emphasize boundaries and bitwise operations used to extract the edge boundaries. On the real segmented image, morphological closing was performed to fill in small holes. To detect discrepancies, a mask was computed by performing a bitwise **AND** between the closed real image and the negation of the dilated Blender image. This mask was then converted to RGB and thresholded to create a binary mask. Finally this binary mask was applied to the original real images and the corresponding pixels were set to zero. Examples of this process are provided in Section 4.2.

4. Results

4.1. Machine Learning

Table 1 presents a comparison of trained models across various performance metrics. Despite being a more modern architecture, the YOLOv4 model did not perform adequately when trained from scratch. Fine-tuning YOLOv4 on COCO weights also failed to produce satisfactory results. In contrast, fine-tuning YOLOv2 pre-trained by Obico [12], a model specifically tailored for stringing detection, produced significantly better results. This demonstrates that leveraging a pre-trained model designed for the specific task can yield superior performance. Unfortunately, further evaluation or tuning of other models reported in the literature was not possible due to the lack of publicly available implementations provided by their respective authors.

Table 1. Performance comparison of trained models. Best results are indicated in bold.

Model	mAP50	Recall	Precision	F1-Score
YOLOv2 (obico)	0.0447	0.07	0.07	0.07
YOLOv2 (fine-tuned)	**0.4152**	**0.50**	**0.64**	**0.56**
YOLOv4 (scratch)	0.1415	0.19	0.30	0.23
YOLOv4 (fine-tuned)	0.1766	0.21	0.32	0.25

Nevertheless, based on the performance metrics, the fine-tuned YOLOv2 model may still indicate generally subpar results. These metrics are sensitive to non-maximum suppression and mainly reflect the accuracy of bounding box predictions. Significant overlap among bounding boxes in many true positive cases negatively impacts these metrics. However, the primary objective of this study was to detect the presence or absence of stringing defects in frames captured during the 3D printing process, to enable timely intervention, thereby reducing material waste and lowering costs.

Qualitative evaluations show that while the model may occasionally miss some stringing instances (resulting in lower recall), it is highly effective at identifying stringing in most images. As evident from the confusion matrix (Figure 6), the model achieves a high detection rate for stringing. Additionally, the false positive count remains very low, highlighting the model's reliability in accurately identifying stringing instances.

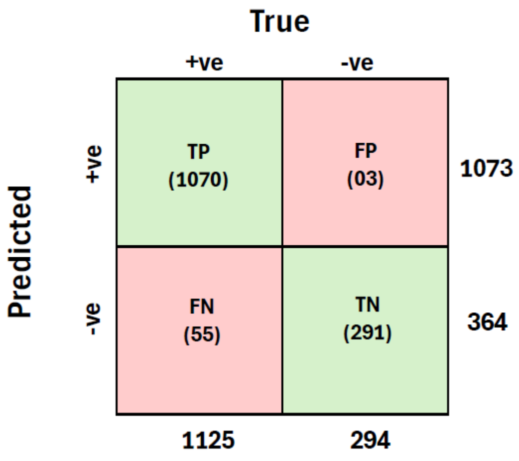


Figure 6. Confusion matrix of the fine-tuned YOLOv2 model. Positive samples represent images with the occurrence of stringing, while negative samples represent error-free images. TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives.

Figure 7 illustrates certain examples of bounding boxes predicted by the original YOLOv2 model from Obico, while Figure 8 illustrates examples of the detections by the fine-tuned YOLOv2 model.

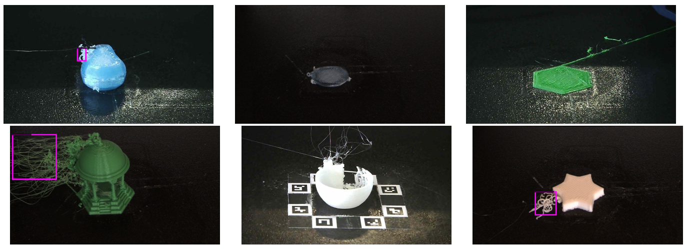


Figure 7. Examples of bounding boxes predicted by the YOLOv2 model from Obico.

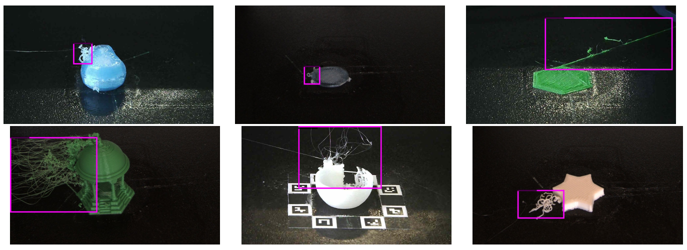


Figure 8. Examples of bounding boxes predicted by the fine-tuned YOLOv2 model.

Furthermore, we believe that the choice of annotation strategy had a significant impact on model performance. Initially, following the approach of Paraskevoudis et al. [11], we used smaller bounding boxes to capture isolated regions of even large, continuous stringing defects, as shown in Figure 9. However, this method produced sub-optimal results, making it challenging for the model to generalize error patterns effectively. As a result, we observed a noticeable decrease in precision. Consequently, we shifted to using larger bounding boxes that encompassed the entirety of the stringing defect, leading to notable improvements in detection accuracy and overall model performance.

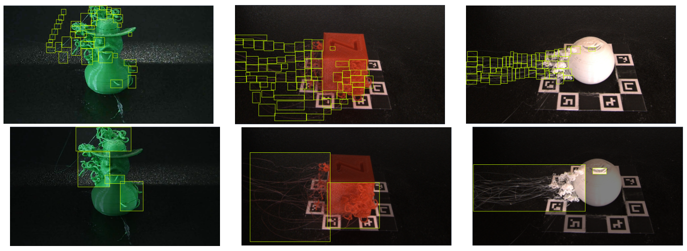


Figure 9. Examples of different annotation strategies for detecting stringing defects. Larger bounding boxes that encompass the entire stringing defect (**bottom**), as opposed to small bounding boxes capturing isolated sections of continuous stringing (**top**).

The inference speed for the YOLOv2 model is about 2–3 FPS and about 1–2 FPS for YOLOv4. With an average print speed of approximately 30 mm/s and a consistent 7-s time lapse between layers, these detection speeds align well with the requirements of the printing process. Frame processing intervals 333–500 ms ensure that each frame is analyzed before the next layer is completed. These findings confirm that the models are capable of detecting stringing in real-time.

4.2. Computer Vision

The method was evaluated on multiple sequences, with the images in Figure 10 illustrating how the G-code is rendered in Blender after adjusting the scene and calibrating the camera.

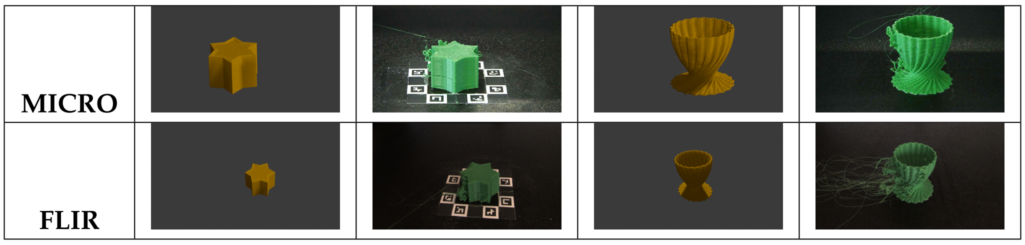
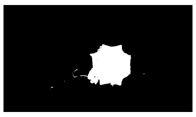
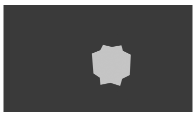
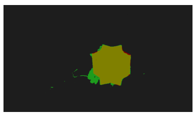
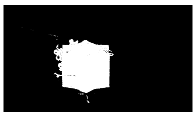
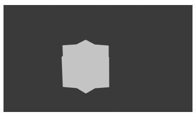
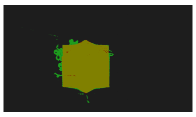
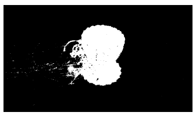
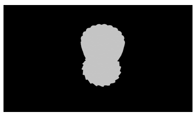
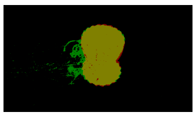
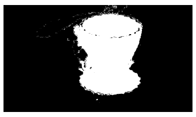
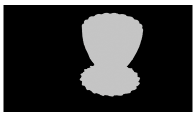
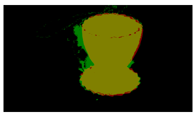


Figure 10. G-code Visualization: Comparisons of Blender Rendering and Real-World Capture.

To visually confirm whether the two images were well-aligned, one image was overlaid on top of the other to ensure alignment, as explained in Section 3.2.4. The results are illustrated in Table 2.

Table 2. Image Comparison: Real and Blender Renderings across Multiple Sequences for FLIR Camera and Microscope.

Real Image Segmented	Blender Image Segmented	Alignment Visualization
		
		
		
		

From the alignment visualization, it was observed that both the real and Blender cameras are well-calibrated, so the object appears from the same position and orientation in both images. This successful alignment was achieved using eight ArUco markers. When attempting to use only three markers, the object from Blender did not match the position and orientation of the real one as demonstrated in Figure 11.

A Monte Carlo simulation was implemented to evaluate the sensitivity of the camera calibration process, and to determine the minimum number of markers required for accurate calibration. For this evaluation, Gaussian noise was added to the position of each marker in the images, with zero mean and 1 pixel standard deviation. This level of noise is representative of the uncertainty associated with the output of the ArUco marker detector. Multiple iterations (1000) were performed of solving the Perspective-n-Point (PnP) problem to compute the camera's location from the noisy markers. The camera locations were saved in each iteration, and this process was repeated for different numbers of markers, ranging from 3 to 8. The plot in Table 3 shows the distribution of the camera positions in both the horizontal (XY) plane and in the vertical (YZ) plane.

As can be seen, it is recommended to use a minimum set of six ArUco markers. Using fewer markers may result in highly uncertain camera locations. In some cases, when only three or four markers are used, the distribution of the camera location often forms two or more clusters, suggesting multiple or under-constrained solutions, even for the low amount of noise used in this study.

The approach for highlighting discrepancies relies on morphological operations. First, a buffer area is defined around the edges of the object in the Blender images where discrepancies between the Blender and real images are ignored. This step prevents false positives in stringing detection caused by minor misalignments between the two images. The process involves applying dilation followed by erosion, with the resulting contours representing areas added during dilation but not removed during erosion. These contours are shown in Figure 12.

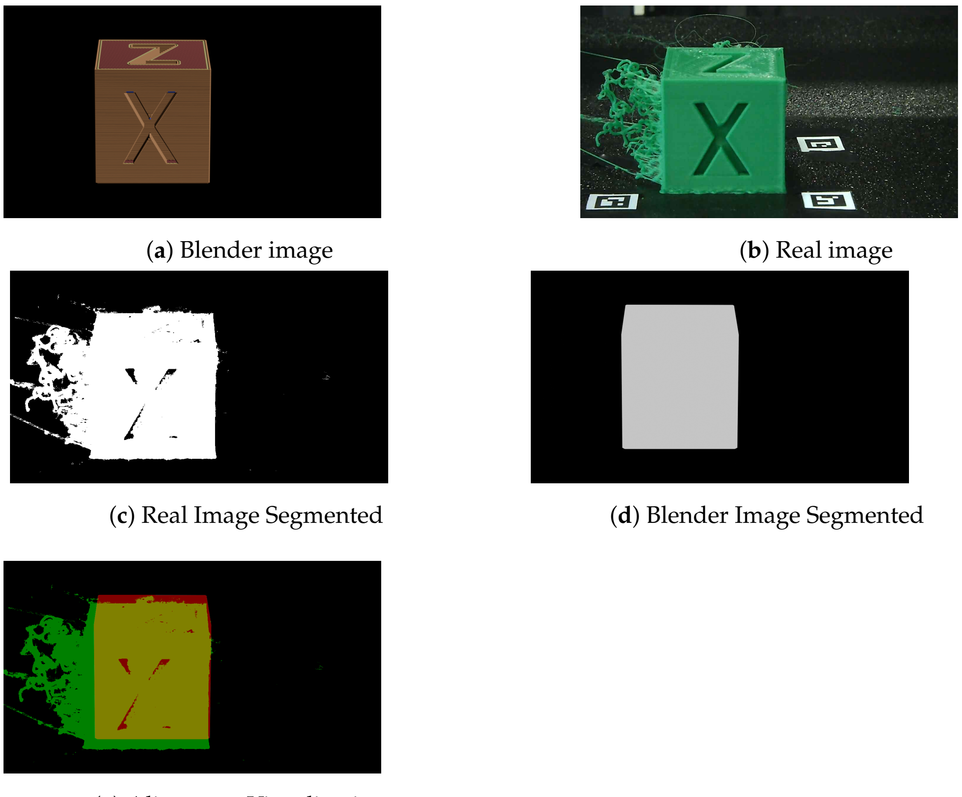

(**a**) Blender image (**b**) Real image (**c**) Real Image Segmented (**d**) Blender Image Segmented (**e**) Alignment Visualization

Figure 11. Image comparison using 3 aruco marker.

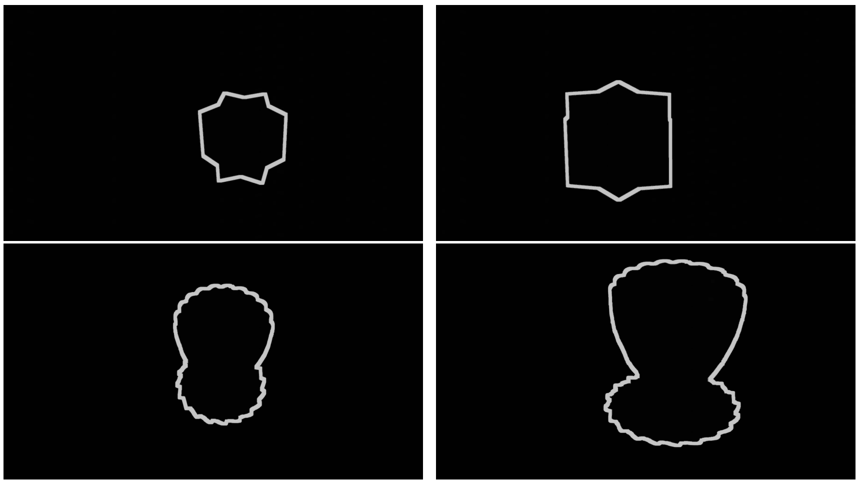

(**a**) FLIR Camera (**b**) Microscope

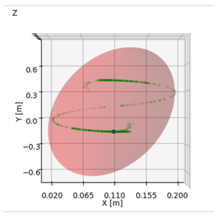
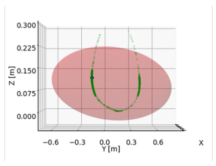
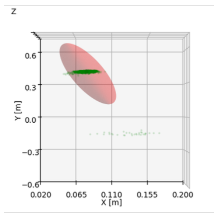
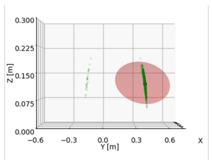
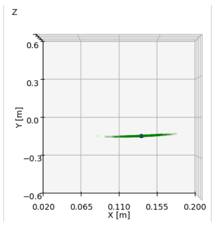
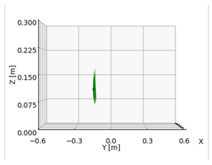
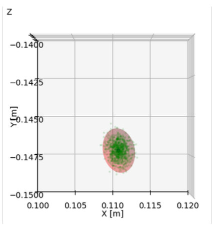
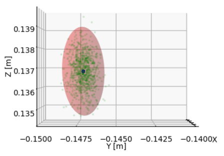
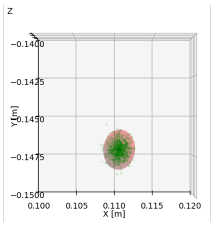
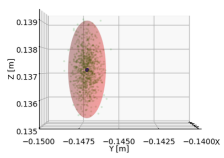
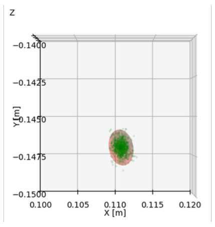
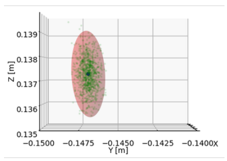
Figure 12. Extracted Edges from Blender Image.

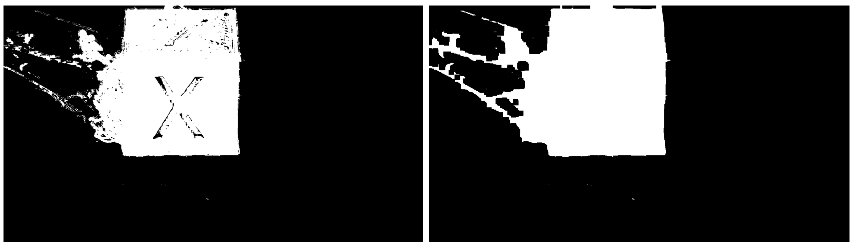
The color-based segmentation of the camera images may produce small errors in areas of the model that are poorly illuminated, such as inside the 'X' marking in Figure 11b. To solve these issues, the closing operation is applied to the segmented area, as shown in Figure 13. This operation closes small holes and connects the disjoint areas.

Finally, as explained in Section 3.2.4. the computed mask was applied to the real segmented images (see Figure 14) and then to the real images (see Figure 15). The result clearly indicates the areas where the real images differs from the Blender ones. It can be seen that these areas of difference correspond to stringing. A simple threshold on the size of the area of difference is sufficient to trigger a warning of printing error due to stringing.

The method performs well for its intended objective of detecting the accumulation of stringing. However, detecting individual filaments is a challenging task. This difficulty arises because the pixels representing single strands often contain a mix of the filament's color, the background color, and the illuminant color. Additionally, this mixture is influenced by factors such as the quality of the optics, the accuracy of the camera's focus, and its depth of field. Using a black background and white illumination helps to mitigate this issue, as this primarily affects the pixel intensity (i.e., the Value channel in the HSV color space) rather than the color itself (i.e., the Hue channel). Empirical tests have shown that filament strands with a width of at least 1–2 pixels in the image are generally well detected.

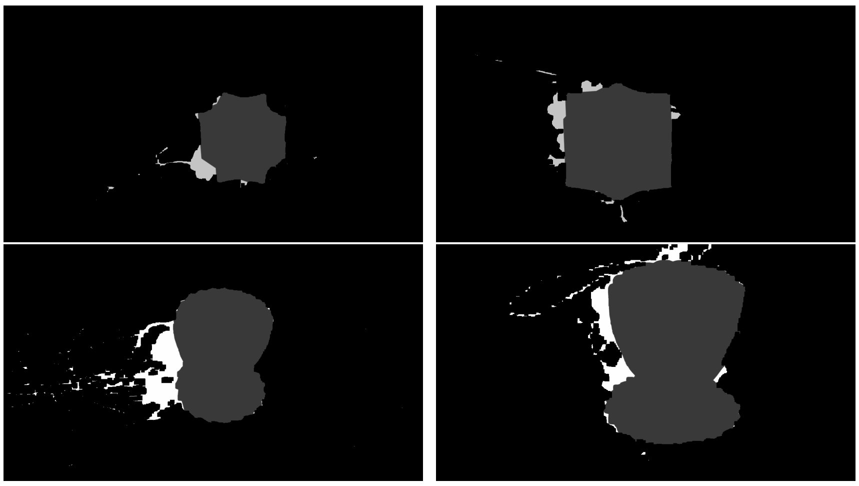
Table 3. Distribution of Camera Centers Using Monte Carlo Simulation: 95% Confidence Ellipsoid (Red), Camera Positions with Noise (Green), Original Camera Locations (Blue).

Number of Markers	Top View	Side View
3		
4		
5		
6		
7		
8		



(**a**) Errors in segmentation (**b**) Closing operation

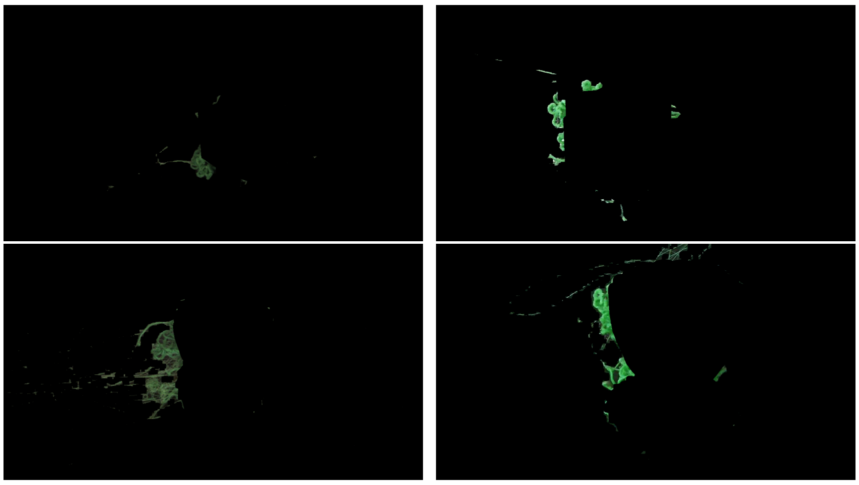
Figure 13. Image processing techniques comparison.



(**a**) FLIR Camera (**b**) Microscope

Figure 14. Masked segmented images.

Petsiuk and Pearce [21] and Charalampous et al. [16] employed similar approaches. However, the method presented here was chosen for several reasons. First, we utilize two cameras, a more cost-effective solution compared to Charalampous et al. , who employ an expensive laser scanner. Second, we visualize the G-code in Blender, thereby avoiding the complexity of using Principled and Translucent BSDF shaders to simulate the appearance of real printed parts. Third, focusing on binary mask images for comparison simplifies the rendering process and reduces computational overhead while still ensuring adequate accuracy for defect detection. This decision was made because the final comparison relies on binary mask images, minimizing the need for photo-realistic rendering.



(**a**) FLIR Camera (**b**) Microscope

Figure 15. Masked Real Image.

Furthermore, the camera images used here were analyzed from the same viewpoint as the synthetic ones. In contrast, the method of Petsiuk and Pearce involved using a projective transformation to convert coordinates from the active printing plane to a virtual top-down view of the printing bed. While this approach allows for orthogonal views, it introduces additional complexity and potential transformation inaccuracies, which were to be avoided. Similarly, Charalampous et al. [16] involved a comparison between the real image with the targeted one, but using point cloud data, which increases computational overhead. The method presented here ensures a direct, consistent, and straightforward comparison between simulated and real images, which aligns with the goal of maximizing reliability in defect detection. Conversely, the process of Petsiuk and Pearce involves comparing a real printed image with a reference "ideal" image to detect defects based on texture analysis using HOG. While effective, morphological operations were preferred for their superior edge detection and segmentation capabilities. By extracting edges in simulated Blender images and enhancing real image segmentation, we create a robust method for identifying discrepancies. This approach simplifies the workflow and improves accuracy. After edge extraction, a mask is computed and applied to the real image, identifying any area outside the mask as an error.

Another method developed by Petsiuk and Pearce [17] relies on computer vision techniques, specifically ICP and multi-template matching. While their results are promising for automated defect mitigation, the authors acknowledged a limitation that their technique may struggle to interpret complex geometries, leading to ambiguous layer contour analysis. In contrast, our masking approach provides a direct and intuitive way of detecting defects, as it highlights only the regions where discrepancies exist between the simulated and real parts.

5. Conclusions

This work explored the possibilities of automatic fault detection in additive manufacturing, focusing on the detection of stringing—a common defect that impacts surface quality and precision, aiming to enhance the quality and reliability of the 3D printing process.

A comprehensive dataset was assembled through a custom data collection setup that integrated a microscope, an industrial inspection camera camera, and time-lapse photography. This dataset formed the foundation for testing machine learning models and computer vision techniques as independent approaches to quality control.

Two approaches were presented and tested. The machine learning approach was designed to recognize stringing patterns based on the dataset, allowing for a proactive approach to defect detection in real-time during the printing process. In parallel, the computer vision method offered real-time monitoring by comparing live images from the printing process with simulated baselines. Aligning the Blender scene with the real-world setup was a critical step, that ensures that the real and simulated images captured the same viewpoint, thus allowing for consistent detection of printing errors. It should be noted that the computer vision method, as implemented, is not intended to distinguish between different types of errors. Such distinction could logically be done, as future work, with a machine learning model such as the one used on the other method, after proper training.

This paper demonstrated that both machine learning and computer vision provide valuable tools for detecting stringing. Machine learning excels in recognizing patterns from large datasets and offers the ability to adapt online, while computer vision enables continuous monitoring and direct comparisons with the expected results. This integration reduces reliance on manual inspection, making quality control in additive manufacturing more efficient. The advancements made through these methods lay a solid foundation for addressing quality issues in additive manufacturing thus contributing to achieving higher-quality prints, minimizing material waste, and enhancing the overall reliability of the 3D printing process.

Author Contributions: Conceptualization, M.D.-E.; methodology, N.G. and H.R; software, O.C.; data collection, O.C.; writing—original draft preparation, O.C.; writing—review and editing, N.G., H.R., I.F.R. and M.D.-E.; supervision, N.G., H.R., I.F.R. and M.D.-E.; funding acquisition, N.G and M.D.-E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets are publicly available on Zenodo (https://doi.org/10.5281/zenodo.14711320).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rajaguru, K.; Karthikeyan, T.; Vijayan, V. Additive manufacturing–State of art. *Mater. Today Proc.* **2020**, *21*, 628–633. [CrossRef]
2. Mukhtarkhanov, M.; Shehab, E.; Ali, M.H. Experimental Study on Warpage Phenomenon of Wax Parts Manufactured by Fused Filament Fabrication. *Polymers* **2024**, *16*, 208. [CrossRef] [PubMed]
3. Kelvin Mark, V.; Raashika, R.; Pradeep Kumar, J.; Arun Prakash, R. Experimental Analysis of the Stringing Problem in FDM Printers. In Proceedings of the International Conference on Advancements in Materials, Design and Manufacturing for Sustainable Development, ICAMDMS 2024, Coimbatore, Tamil Nadu, India, 23–24 February 2024. [CrossRef]
4. Lee, W.; Fritsch, J.; Maqsood, A.; Liu, S.; Bourassa, T.; Calara, R.; Kim, W.S. Adaptive 3D printing for in situ adjustment of mechanical properties. *Adv. Intell. Syst.* **2023**, *5*, 2200229. [CrossRef]
5. Kayali, Y.; Ding, M.; Hamdallah, S.; Qi, S.; Bibb, R.; Gleadall, A. Effect of printing parameters on microscale geometry for 3D printed lattice structures. *Mater. Today Proc.* **2022**, *70*, 31–37. [CrossRef]
6. Simplify3D. Print Quality Troubleshooting Guide. Available online: https://www.simplify3d.com/resources/print-quality-troubleshooting/ (accessed on 4 May 2024).
7. Delli, U.; Chang, S. Automated process monitoring in 3D printing using supervised machine learning. *Procedia Manuf.* **2018**, *26*, 865–870. [CrossRef]
8. Yean, F.P.; Chew, W.J. Detection of Spaghetti and Stringing Failure in 3D Printing. In Proceedings of the 2024 International Conference on Green Energy, Computing and Sustainable Technology (GECOST), Virtual, 17–19 January 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 293–298. [CrossRef]
9. Karna, N.B.A.; Putra, M.A.P.; Rachmawati, S.M.; Abisado, M.; Sampedro, G.A. Toward accurate fused deposition modeling 3D printer fault detection using improved YOLOv8 with hyperparameter optimization. *IEEE Access* **2023**, *11*, 74251–74262. [CrossRef]
10. Jocher, G.; Qiu, J.; Chaurasia, A. Ultralytics YOLO [Software]. Available online: https://github.com/ultralytics/ultralytics (accessed on 15 February 2024)).
11. Paraskevoudis, K.; Karayannis, P.; Koumoulos, E.P. Real-time 3D printing remote defect detection (stringing) with computer vision and artificial intelligence. *Processes* **2020**, *8*, 1464. [CrossRef]
12. Obico. The Spaghetti Detective [Software]. 2022. Available online: https://github.com/TheSpaghettiDetective/obico-server (accessed on 12 March 2024).
13. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6517–6525. [CrossRef]
14. Redmon, J. Darknet: Open Source Neural Networks in C. 2016. Available online: http://pjreddie.com/darknet/ (accessed on 15 March 2024).
15. AbouelNour, Y.; Gupta, N. Assisted defect detection by in-process monitoring of additive manufacturing using optical imaging and infrared thermography. *Addit. Manuf.* **2023**, *67*, 103483. [CrossRef]
16. Charalampous, P.; Kostavelis, I.; Kopsacheilis, C.; Tzovaras, D. Vision-based real-time monitoring of extrusion additive manufacturing processes for automatic manufacturing error detection. *Int. J. Adv. Manuf. Technol.* **2021**, *115*, 3859–3872. [CrossRef]
17. Petsiuk, A.; Pearce, J.M. Open source computer vision-based layer-wise 3D printing analysis. *Addit. Manuf.* **2020**, *36*, 101473. [CrossRef]
18. Thomas, L.; Gehrig, J. Multi-template matching: A versatile tool for object-localization in microscopy images. *BMC Bioinform.* **2020**, *21*, 1–8. [CrossRef] [PubMed]

19. Besl, P.J.; McKay, N.D. Method for registration of 3-D shapes. In Proceedings of the Sensor Fusion IV: Control Paradigms and Data Structures, SPIE, Boston, MA, USA, 12–15 November1992; Volume 1611, pp. 586–606. [CrossRef]
20. Baumann, F.; Roller, D. Vision based error detection for 3D printing processes. In Proceedings of the MATEC Web of Conferences. EDP Sciences, Amsterdam, The Netherlands, 23–25 March 2016; Volume 59, p. 06003. [CrossRef]
21. Petsiuk, A.; Pearce, J.M. Towards smart monitored AM: Open source in-situ layer-wise 3D printing image anomaly detection using histograms of oriented gradients and a physics-based rendering engine. *Addit. Manuf.* **2022**, *52*, 102690. [CrossRef]
22. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–26 June 2005; Volume 1, pp. 886–893.
23. Roboflow, Inc. Roboflow [Software]. Available online: https://roboflow.com/ (accessed on 26 July 2024).
24. Google Research. Google Colaboratory. Available online: https://colab.google/ (accessed on 15 February 2024).
25. Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F.J.; Marín-Jiménez, M.J. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognit.* **2014**, *47*, 2280–2292. [CrossRef]
26. Ayyappan, S. Import-G-Code [Software]. Available online: https://github.com/blender-for-science/import-G-code (accessed on 1 April 2024).
27. Denninger, M.; Sundermeyer, M.; Winkelbauer, D.; Zidan, Y.; Olefir, D.; Elbadrawy, M.; Lodhi, A.; Katam, H. Blenderproc. *arXiv* **2019**, arXiv:1911.01911.