engineering proceedings

MDPI

Proceeding Paper

Machine Learning G-Code Optimization †

Héctor Lasluisa-Naranjo [1,*], David Rivas-Lalaleo [2], Joaquín Vaquero-López [1] and Christian Cruz-Moposita [3]

[1] International Doctoral School, Universidad Rey Juan Carlos, 28933 Madrid, Spain; joaquin.vaquero@urjc.es
[2] Department of Electronics and Telecommunications, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador; drrivas@espe.edu.ec
[3] DIDAE Aeronautical Research and Development Department, Ejército Ecuatoriano, 170403 Quito, Ecuador; crcruzm@ejercitoecuatoriano.mil.ec
* Correspondence: h.lasluisa.2022@alumnos.urjc.es
† Presented at the XXXII Conference on Electrical and Electronic Engineering, Quito, Ecuador, 12–15 November 2024.

Abstract: G-codes are essential in CNC systems, providing crucial instructions for controlling machine parameters and operations in manufacturing, including 3D printing. They may contain errors affecting product quality and increasing resource consumption. This research applies the K-means machine learning clustering algorithm to optimize G-code parameters such as extruder and heated bed temperature, deposition speed, and flow control. The objective is to reduce manufacturing time and material usage while maintaining surface quality. A line-by-line analysis and rewriting of the G-code resulted in an average 24.36% reduction in time and 5% in material use, with minimal impact on quality, validated with the Taguchi method.

Keywords: G-code; machine learning; 3D printing

1. Introduction

G-codes, which are fundamental to computerized numerical control (CNC) systems, play a critical role in both machining and 3D printing. These commands provide instructions to control the motion and actions of CNC machines during manufacturing [1]. In addition, they play a key role in 3D printing, where the trajectory is controlled by these codes, allowing the quality of printed products to be evaluated through image analysis and result comparisons [2]. Understanding G-codes is essential for CNC programming, as they dictate parameters such as cutting feed, depth, and speed, and they are the foundation of modern manufacturing processes [3].

In 3D printing, G-codes control critical parameters such as trajectories, motion speed, fill patterns, material flow, and temperature [4]. Problems with G-codes originate at the CAD design stage, where NURBS-based programs offer higher quality finishes than those based on parametric curves and polygonal approximations [5]. G-code errors can also be introduced during the conversion of 3D models to STL format, creating gaps, holes, and open loops in cross sections that affect manufacturing quality [6]. In addition, another source of G-code errors comes from the wide variety of 3D printing equipment brands and slicing software, each with its own idiosyncrasies and standards [7].

Machine learning (ML), a branch of artificial intelligence (AI), is increasingly being used to optimize G-code. Neural networks efficiently transform numerically controlled G-codes [8]. AI algorithms optimize G-codes in CNC machines, reducing paths and improving processes (e.g., PCB drilling) [9]. These advances promise greater efficiency and accuracy in programming and machining. The best practices for debugging G-codes include statistical models and ML to detect malicious edits without access to original models, thus ensuring the quality of 3D-printed parts [10].

This study proposes the optimization of G-codes in 3D printing using the K-means algorithm, which is widely used in unsupervised clustering to evaluate parameter variation

check for updates

Citation: Lasluisa-Naranjo, H.; Rivas-Lalaleo, D.; Vaquero-López, J.; Cruz-Moposita, C. Machine Learning G-Code Optimization. *Eng. Proc.* **1970**, 77, 32. https://doi.org/10.3390/engproc2024077032

Academic Editor: Christian Tipantuña

Published: 18 November 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

among objects [11]. In the context of 3D printing, K-means is applied to the modular partitioning of FDM 3D printers to improve performance, functionality, and product cost by generalizing modules [12].

In addition, K-means is used to predict uncertain material parameters when updating finite element models in 3D printing by optimizing an objective function based on dynamic characteristics to determine material properties [13]. The algorithm also improves the efficiency of 3D printing processes by determining optimal initial centroids, reducing iterations, and execution time [14]. It is also used to select interpolation points in the ISDF method, reducing computational cost and memory usage [15].

2. Methodology

This section describes the method used to optimize 3D printing parameters, specifically to reduce printing time and material consumption without compromising the quality of the printed model. This was achieved using the K-means algorithm, a machine learning technique that clusters data into specific groups. In the field of artificial intelligence, K-means is known for its ability to efficiently identify patterns and structures in large data sets [16]. This algorithm is particularly suitable for the present study due to its computational efficiency, its ability to identify patterns and trends in pressure parameters, and its versatility in addressing clustering problems in various contexts [17].

2.1. Experimental Model Selection

The 3Dbenchy model [18] was selected for the development of the 3D printing tests. This popular benchmark is widely used in the maker community as a standard for evaluating printer performance. The 3Dbenchy model presents multiple printing challenges, including changes in layer height, speed, and temperature, making it ideal for testing parameter optimization. Using the 3Dbenchy allows for a comprehensive evaluation of how the optimized parameters affect print quality, build time, and material consumption. Figure 1 shows the 3Dbenchy CAD model.

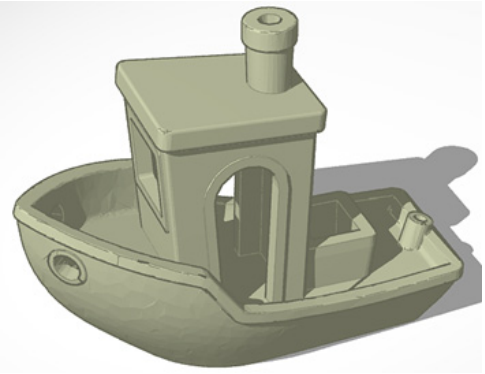


Figure 1. Three-dimensionalbenchy CAD model.

2.2. Slicing Software

To generate the input G-code file and evaluate the effectiveness and applicability of the algorithm, three different slicers were employed: Cura, PrusaSlicer, and IdeaMaker. Each slicer possesses distinctive features, path generation algorithms, and configurations, as illustrated in Table 1, allowing for a comprehensive and diverse comparison of the features.

Table 1. Main features of slicing software.

	Cura	Prusa Slicer	Ideamaker
Developer	Ultimaker	Alessandro Ranellucci	Raise 3D
Algorithm	Arachne Engine	Slic3r	Pathfinder
Principle	Voronoi Diagram	Planar Intersection, tessellación	Algorithms A star Dijkstra
Disadvantage	Unnecessary non-extrusion movements and longer printing time.	False retraction movements and other non-essential movements.	Low trajectory optimization

2.3. Coding

The proposed solution begins with the reading of a G-code file, which is provided by the slicing software. The file contains specific codes, including M104, M109, M140, and M190 for temperature control, G1 for speed control, and M221 for flow control. In the ML process, the K-means algorithm employs the input parameters to cluster the G-code data and calculate new values (centroids). The infill density is fixed at 10%, and the infill speed is fixed at 80 mm/s. These new values are then replaced with the relevant lines of code, resulting in the creation of an optimized G-code file. For more detailed information, please refer to Figure 2.

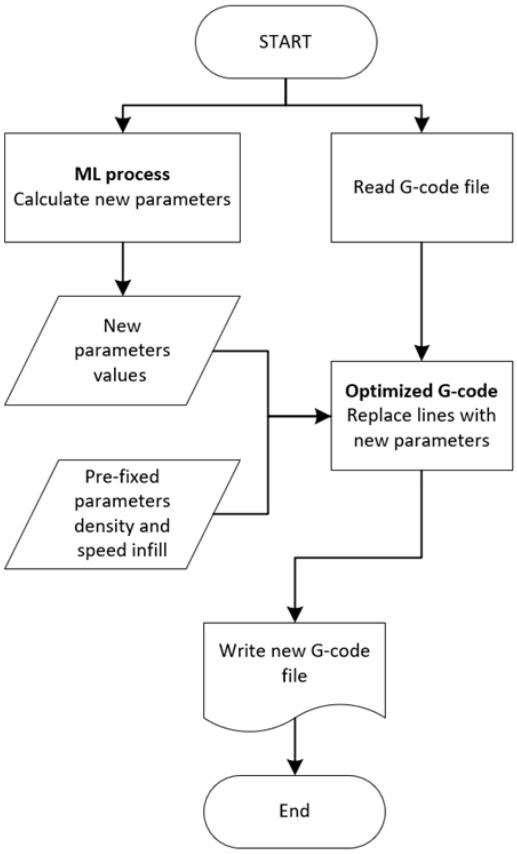


Figure 2. G-code optimizer flowchart.

2.4. Data Preprocessing

For the machine learning process, the data from successful 3D prints were collected, which included essential parameters such as layer height, print speed, first layer speed,

perimeter speed, extruder temperature, bed temperature, and flow rate. These data points are presented in Table 2 for reference.

Table 2. Three-dimensional printing input parameters.

Parameter	Measures				
Layer height (mm)	0.2	0.2	0.3	0.1	0.3
Print speed (mm/s)	50	60	45	55	65
First layer speed (mm/s)	20	30	25	15	35
Perimeter speed (mm/s)	40	50	45	35	55
Extruder Temp (°C)	200	210	195	205	220
Bed Temp (°C)	60	65	55	60	70
Flow rate (%)	100	95	105	90	110

The data were organized into a dictionary and subsequently converted into a DataFrame using the pandas library, allowing for convenient manipulation and analysis.

2.4.1. Library Imports

NumPy (imported as np) and pandas (imported as pd) were imported for data manipulation and mathematical operations, and KMeans from sklearn cluster was imported to apply the K-means clustering algorithm for data grouping.

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
```

2.4.2. Data Definition

Dictionary-named data were defined, where each key represents a 3D printing parameter, and each value is a list of corresponding measurements. Parameters include layer height, print speed, first layer speed, perimeter speed, extruder temperature, bed temperature, and flow rate.

data = {'layer_height': [0.2, 0.2, 0.3, 0.1, 0.25],'print_speed': [50, 60, 45, 55, 65],'first_layer_speed': [20, 30, 25, 15, 35],'perimeter_speed': [40, 50, 45, 35, 55],'extruder_temp': [200, 210, 195, 205, 220],'bed_temp': [60, 65, 55, 60, 70],'flow_rate': [100, 95, 105, 90, 110]}.

2.4.3. DataFrame Creation

A DataFrame df is created using pandas from the data dictionary, organizing the data into a tabular format where each column corresponds to a parameter and each row to an observation.

```
df = pd.DataFrame(data)
```

2.5. K-Means Model Training

2.5.1. K-Means Algorithm Initialization

An instance of the K-means clustering algorithm from the sklearn library is created with *n*_clusters = 5, specifying that the data should be grouped into 5 clusters. This parameter can be adjusted based on the analysis goals.

```
kmeans = KMeans(n_clusters = 5)
```

2.5.2. Model Fitting

The fit() method was used to train the K-means model on the data contained in the DataFrame df. During this process, the algorithm clusters the data based on the features provided.

```
kmeans.fit(df)
```

2.5.3. Assigning Cluster Labels

After fitting the model, the labels_ attribute of the K-means object contain the cluster labels for each observation in the DataFrame. These labels indicate the cluster to which each row belongs. A new column, 'cluster', was added to the DataFrame to store these cluster labels.

```
df['cluster'] = kmeans.labels_
```

2.6. Optimal Parameter Selection

2.6.1. Identifying the Most Frequent Cluster

df['cluster'].value_counts() counted the occurrences of each cluster label in the 'cluster' column of the DataFrame. The value_counts() method returned a pandas Series with counts in descending order. idxmax() retrieved the index of the maximum value in the Series, which corresponds to the most frequent cluster label. This result was stored in the variable optimal_cluster, indicating the most common cluster.

```
optimal_cluster = df['cluster'].value_counts().idxmax()
```

2.6.2. Calculating Average Parameters for the Optimal Cluster

- df[df['cluster'] == optimal_cluster] filters the DataFrame to select only the rows belonging to the most frequent cluster (optimal_cluster).
- drop(['cluster'], axis = 1) removes the 'cluster' column from the filtered rows as it is no longer needed for the average parameter calculation.
- mean() computes the mean of each column in the filtered DataFrame, resulting in a pandas Series with average values for each parameter.
- to_dict() converts this Series into a dictionary where the keys are parameter names and the values are the computed averages. The result is stored in the variable optimized_params.

```
optimized_params = df[df['cluster'] == optimal_cluster].drop(['cluster'], axis = 1).mean().to_dict()
```

Additionally, the printing parameters were set to an infill speed of 80 mm/s and an infill density of 10%.

```
infill_speed = 80
infill_density = 10
```

2.7. G-Code Optimization

2.7.1. Import the Regular Expression Module

The re module was imported, which provides functions for working with regular expressions in Python. This step is useful for performing search and replace operations in the text.

```
import re
```

2.7.2. Define the Function

The function optimize_gcode was defined to take four arguments: input_file_path (path to the input G-code file), output_file_path (path to the optimized G-code file), params (a dictionary containing printing parameters), infill_speed (speed of infill), and infill_density (density of infill).

```
def optimize_gcode(input_file_path, output_file_path, params, infill_speed, infill_density):
```

2.7.3. Read the Input File

The input file was opened in read mode ('r') and all lines were read into a list called lines.

```
with open(input_file_path, 'r') as file:
lines = file.readlines()
```

2.7.4. Initialize the List of Optimized Lines

An empty list was created called optimized_lines to store the modified lines of G-code.

```
optimized_lines = []
```

2.7.5. Process Each Line to Apply Optimizations

Each line in the input G-code file was iterated to apply the required modifications based on the content and parameters provided.

```
for line in lines:
```

2.7.6. Update Layer Height Comment

If the line started with; (indicating a comment) and contained 'Layer height', the comment was updated to reflect the layer height specified in params.

```
if line.startswith(';') and 'Layer height' in line:
    line = f';Layer height: {params["layer_height"]}\n'
```

2.7.7. Adjust Print Speed

For lines containing G1, F (indicating speed), and not starting with ;, replaced the speed value (F) with the speed specified in params, converted from mm/s to mm/min.

```
if 'G1' in line and 'F' in line and not ';' in line.split('G1')[0]:
    line = re.sub(r'(F)(\d+)', lambda m: f'F{int(params["print_speed"] * 60)}', line)
```

2.7.8. Adjust First Layer Speed

This step is similar to print speed adjustment but applied to lines containing G1, Z (indicating layer height), and F, modifying the speed for the first layer.

```
if 'G1' in line and 'Z' in line and 'F' in line and not ';' in line.split('G1')[0]:
    line = re.sub(r'(F)(\d+)', lambda m: f'F{int(params["first_layer_speed"] * 60)}', line)
```

2.7.9. Adjust Perimeter Speeds

We updated the speeds for outer and inner wall types, replacing F with the perimeter speed specified in params.

```
if ';TYPE:WALL-OUTER' in line or ';TYPE:WALL-INNER' in line:
line = re.sub(r'(F)(\d+)', lambda m: f'F{int(params["perimeter_speed"] * 60)}', line)
```

2.7.10. Adjust Infill Speed

For lines containing;TYPE:FILL, the speed F was replaced with the specified infill_speed.

```
elif ';TYPE:FILL' in line:
    line = re.sub(r'(F)(\d+)', lambda m: f'F{int(infill_speed * 60)}', line)
```

2.7.11. Modify Extruder and Bed Temperatures

The extruder and bed temperatures were adjusted by updating lines containing M104, M109, M140, or M190 with the temperatures specified in params.

```
if 'M104' in line or 'M109' in line:

    line = f'M104 S{int(params["extruder_temp"])}\n' if 'M104' in line else f'M109 S{int
(params["extruder_temp"])}\n'
    if 'M140' in line or 'M190' in line:
```

line = f'M140 S{int(params["bed_temp"])}\n' if 'M140' in line else f'M190 S{int(params["bed_temp"])}\n'

2.7.12. Adjust Material Flow Rate

For lines containing G1 and E (indicating material flow), the flow value E is adjusted based on the specified flow_rate in params.

if 'G1' in line and 'E' in line and not ';' in line.split('G1')[0]:
line = re.sub(r'(E)(\d+\.?\d*)', lambda m: f'E{float(m.group(2)) * params["flow_rate"] / 100}', line)

2.7.13. Update Infill Density

Lines containing M221were modified to set the infill density using the provided infill_density. The function optimized_lines.append(line) guaranteed that all lines, both modified and unmodified, are incorporated into the final optimized G-code file.

if 'M221' in line:
line = f'M221 S{int(infill_density)}\n'
optimized_lines.append(line)

2.7.14. Write to Output File

Finally, the output file was opened in write mode ('w') and all the optimized lines were written to this file.

with open(output_file_path, 'w') as file:
file.writelines(optimized_lines)

2.8. Deployment of Optimization Function

The paths for the original and optimized G-code files were defined. The optimize_gcode function was then called with the optimized parameters, including the values set for infill speed and density. This function generated the optimized G-code file.

input_file_path = r"file_path"
output_file_path = r"file_path"
optimize_gcode(input_file_path, output_file_path, optimized_params, infill_speed, infill_density)

The input_file_path and output_file_path variables were assigned the respective paths to the source and destination G-code files. The prefix r indicates that these strings were raw literals, ensuring that backslashes were handled correctly.

This function call executed optimize_gcode, passing:

- input_file_path: the location of the G-code file to be optimized.
- output_file_path: the destination path for the optimized G-code file.
- optimized_params: a dictionary of parameters including layer height, print speed, first layer speed, perimeter speed, extruder temperature, bed temperature, and material flow rate.
- infill_speed: the speed of infill printing, in millimeters per second.
- infill_density: the percentage of infill density to be applied.

In summary, this method enables the optimization of 3D printing parameters through the use of the K-means algorithm, which adjusts the resulting G-code to enhance both the efficiency and quality of the printing process.

3. Results and Discussion

With the three new optimized G-code files, the 3D benchy model was 3D printed in PLA material, as shown in Figure 3.



(**a**) (**b**) (**c**)

Figure 3. Models printed with optimized G-code based on slicers (**a**) Cura, (**b**) Prusa Slicer, (**c**) Idea Maker.

Despite having undergone optimization, the models displayed a multitude of characteristics and surface details, which can be attributed primarily to the slicer software utilized. Table 3 presents a comparative analysis of the features between the optimized models and those that have not undergone optimization. The results of the percentage variation in each case are highlighted.

Table 3. Comparison of traditional and optimized 3D printing performance measures.

		Time	Material	Surface Finish	Accuracy
CURA	Original	48 m 14 s	3724 mm	15 µm	99.41%
	Optimized	39 m 33 s	3538 mm	14.8 µm	99.39%
	% variation	**−18%**	**−5%**	**−1%**	**−0.02%**
PRUSASLICER	Original	1 h 22 m 21 s	4297 mm	16.5 µm	99.4%
	Optimized	51 m 56 s	4082 mm	16.3 µm	99.38%
	% variation	**−36.90%**	**−5%**	**−1%**	**−0.01%**
IDEAMAKER	Original	52 m 48 s	3908 mm	18 µm	99.13%
	Optimized	43 m 12 s	3712 mm	17.7 µm	99.12%
	% variation	**−18.18%**	**−5.02%**	**−1.5%**	**−0.01%**

To assess the efficacy of the G-code optimization script, the Taguchi statistical method, a widely utilized approach in experimental design, will be employed. This method will be implemented through an experimental framework, with four levels selected for each factor, based on both the recommendations for optimal FDM printing parameters for PLA [19] and the technical specifications of the Artillery Hornet printer [20]. The values of this configuration are presented in Table 4.

Table 4. Input parameters.

Layer Height (mm)	Extruder Temp (°C)	Bed Temp (°C)	Print Speed (mm/s)	Flow Rate (%)
0.1	195	55	45	90
0.15	200	60	50	95
0.2	205	65	55	100
0.3	210	70	60	105

Results

The XLSTAT software employs the provided data to assign an L16 orthogonal matrix for the experimental design, thereby generating 16 potential parameter combinations.

A test object is produced for each combination, and the resulting output variables are subsequently measured, as illustrated in the results presented in Table 5.

Table 5. Combinations of input parameters and measured printing variables.

Experiment	Layer Height (mm)	Extruder Temp (°C)	Bed Temp (°C)	Print Speed (mm/s)	Flow Rate (%)	Fabrication Time Variation (%)	Material Consumption Variation (%)	Accuracy Variation (%)
1	0.1	195	55	45	90	−12.5	−2.5	−0.5
2	0.1	200	60	50	95	−15	−1.5	−1
3	0.1	205	65	55	100	−17.75	−3	−0.75
4	0.1	210	70	60	105	−20.5	−4.25	−1.5
5	0.15	195	60	55	105	−19	−3.75	−1.25
6	0.15	200	55	60	100	−21.5	−2.25	−1.75
7	0.15	205	70	45	95	−23.25	−4.5	−0.85
8	0.15	210	65	50	90	−22	−3.5	−1
9	0.2	195	65	60	95	**−24**	**−5**	**−0.01**
10	0.2	200	70	55	90	−21	−3	−1.1
11	0.2	205	55	50	105	−19.5	−2.5	−1.4
12	0.2	210	60	45	100	−16.75	−2.75	−1.2
13	0.3	195	70	50	100	−14	−1.75	−1.3
14	0.3	200	65	45	105	−10	−1.25	−1.5
15	0.3	205	60	60	90	−7.5	0.5	−1.75
16	0.3	210	55	55	95	−5	1.25	−1.9

As illustrated in Figure 4, the extruder temperature (195 °C) in experiment 9 is notably lower than in the other experiments, which may have contributed to the prevention of over-extrusion. Conversely, the bed temperature (65 °C) is within the typical range, facilitating optimal adhesion of the initial layers without warping the test object. The layer height (0.2 mm) represents an intermediate value, and in conjunction with the printing speed (60 mm/s), it facilitated a reduction in fabrication time without a significant impact on dimensional accuracy in this experiment.

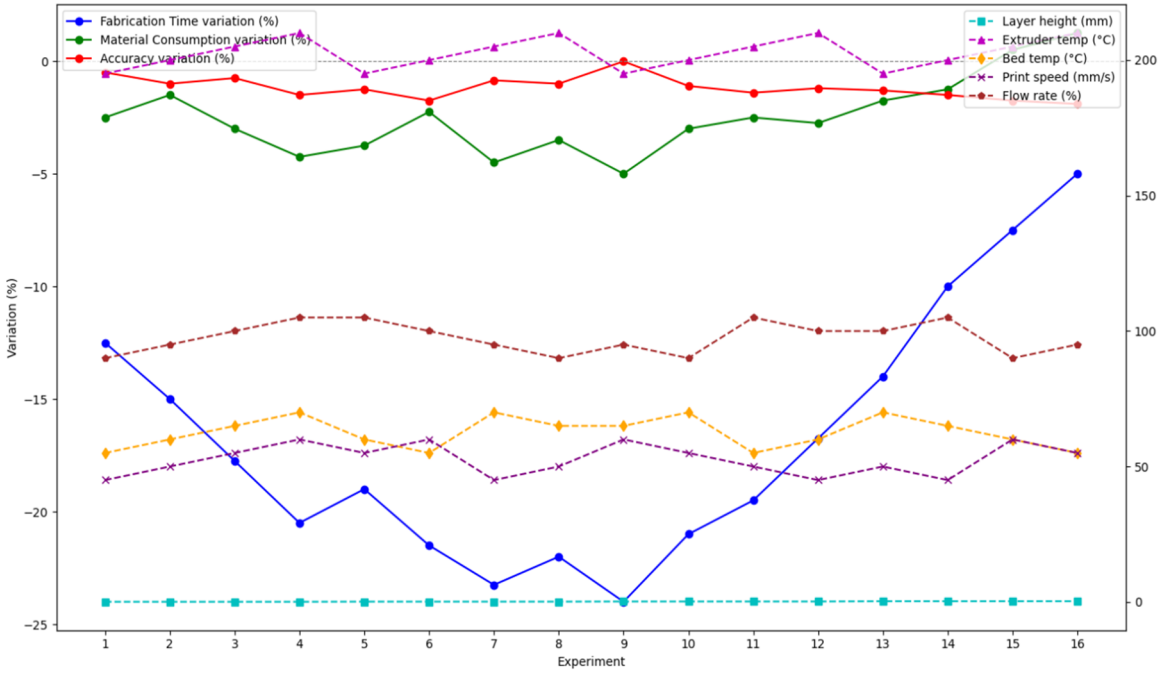


Figure 4. Fabrication time, material consumption, and accuracy variation.

A level of linearity consistent with this manufacturing technology can be observed, which facilitates the identification of general trends and the primary influences of input parameters on output variables. The Taguchi method is designed to enhance efficiency in the search for optimal configurations by limiting the number of experiments required. In this context, the linearity of the data allows for practical inferences to be drawn without the need for an excessive number of trials. A fully nonlinear approach would necessitate a significantly greater number of experiments, which would unnecessarily complicate the subsequent analysis.

4. Conclusions

The analysis using the Taguchi method demonstrated that optimizing 3D printing parameters using the developed code is effective in enhancing manufacturing process efficiency. The optimal combination of parameters successfully reduced print time by an average of 24.36% across all three cases and decreased material consumption by 5%, without compromising the surface quality and accuracy of the printed object.

These findings confirm that, despite the advanced algorithms employed by current slicers (such as Cura with Voronoi diagrams, PrusaSlicer with plane intersection and tessellation, and IdeaMaker with A* and Dijkstra algorithms), there is significant room for improvement through the application of machine learning techniques and robust analysis like the Taguchi method. This additional optimization addresses inherent weaknesses in slicer algorithms, achieving a more efficient and higher-quality 3D printing process.

This study provides a robust methodology for parameter optimization in 3D printing, demonstrating that combining advanced machine learning techniques with experimental design can lead to substantial improvements in manufacturing performance and final product quality.

Author Contributions: Conceptualization, H.L.-N. and D.R.-L.; methodology, J.V.-L. and C.C.-M.; software, C.C.-M.; validation, H.L.-N. and J.V.-L.; formal analysis, H.L.-N.; investigation, H.L.-N. and C.C.-M.; resources, H.L.-N.; data curation, D.R.-L.; writing—original draft preparation, H.L.-N.; writing—review and editing, J.V.-L.; visualization, C.C.-M.; supervision, D.R.-L.; project administration, H.L.-N.; funding acquisition, H.L.-N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: This study did not require ethical approval.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this paper were extracted from practical experiments, are not stored in databases and do not violate any privacy policy.

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

References

1. Daskalogrigorakis, G.; Kirakosian, S.; Marinakis, A.; Nikolidakis, V.; Pateraki, I.; Antoniadis, A.; Mania, K. G-Code Machina: A Serious Game for G-code and CNC Machine Operation Training. In Proceedings of the 2021 IEEE Global Engineering Education Conference (EDUCON), Vienna, Austria, 21–23 April 2021. [CrossRef]
2. Bae, H.C. G-code 3D Method and Apparatus for Managing 3D Printing Using G-code. 2019.
3. Negi, P.K.; Ram, M.; Yadav, O.P. CNC Machine Programming Codes (G-Codes and M-codes). In *Basics of CNC Programming*, 1st ed.; Taylor & Francis: New Dehli, India, 2018. [CrossRef]
4. John, P.; Komma, V.R.; Bhore, S.P. Development of MATLAB Code for Tool Path Data Extraction from the G Code of the Fused Filament Fabrication (FFF) Parts. *Eng. Res. Express* **2023**, *5*, 025018. [CrossRef]
5. Pascual, A.; Ortega, N.; Plaza, S.; Holgado, I.; Arrizubieta, J.I. A RE Methodology to Achieve Accurate Polygon Models and NURBS Surfaces by Applying Different Data Processing Techniques. *Metals* **2020**, *10*, 1508. [CrossRef]
6. Husiev, O.; Nikiforova, T. Research of the Converting Stages for the Volume Model of the Product into the Control Code for a 3D Printer in the Context of Automated Construction of 3D Printing Technology. *Ukr. J. Constr. Archit.* **2022**, *2312*, 250822. [CrossRef]
7. Eva, S.-C.; Sover, A.; Ermolai, V. The Impact of the G-code Flavour Selection in FFF. In Proceedings of the 4th International Conference. Business Meets Technology, Ansbach, Germany, 7–9 July 2022. [CrossRef]

8. Xu, H.; Wu, J.; Qin, Y.; Lin, X.; Song, H.; Zhu, W. Method for Converting Numerically Controlled G-Codes. Patent WO2007078025A1, 12 July 2007.
9. Aciu, R.-M.; Ciocarlie, H. G-code Optimization Algorithm and Its Application on Printed Circuit Board Drilling. In Proceedings of the 2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 15–17 May 2014. [CrossRef]
10. Beckwith, C.; Naicker, H.S.; Mehta, S.; Udupa, V.R.; Nim, N.T.; Gadre, V.; Pearce, H.; Mac, G.; Gupta, N. Needle in a Haystack: Detecting Subtle Malicious Edits to Additive Manufacturing G-Code Files. *IEEE Embed. Syst. Lett.* **2022**, *14*, 111–114. [CrossRef]
11. Li, X.; Tan, H. K-Means Algorithm Based on Initial Cluster Center Optimization. In *Cyber Security Intelligence and Analytics*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12337, pp. 477–485. [CrossRef]
12. You, Y.; Liu, Z.; Liu, Y.; Huang, Y.; Huang, Q. K-Means Module Division Method of FDM 3D Printer-Based Function–Behavior–Structure Mapping. *Appl. Sci.* **2023**, *13*, 7453. [CrossRef]
13. Mathur, M.B. K-means Optimizer: An Efficient Optimization Algorithm for Predicting the Uncertain Material Parameters in Real Structures. In Proceedings of the 5th International Conference on Numerical Modelling in Engineering, Ghent, Belgium, 23–24 August 2022; Lecture Notes in Electrical Engineering. Springer: Singapore, 2023; Volume 833, pp. 95–106. [CrossRef]
14. Qin, X.; Li, J.; Hu, W.; Yang, J. Machine Learning K-Means Clustering Algorithm for Interpolative Separable Density Fitting to Accelerate Hybrid Functional Calculations with Numerical Atomic Orbitals. *J. Phys. Chem. A* **2020**, *124*, 7397–7407. [CrossRef] [PubMed]
15. Vinué, G.; Simó, A.; Alemany, S. The K-means Algorithm for 3D Shapes with an Application to Apparel Design. *Adv. Data Anal. Classif.* **2016**, *10*, 297–312. [CrossRef]
16. Fitri, M.; Wardhani, A.R.; Purnomowati, W.; Vitianingsih, A.V.; Maukar, A.L.; Puspitarini, E.W. Potential Customer Analysis Using K-means with Elbow Method. *JIKO (J. Inform. Dan Komput.)* **2023**, *7*, 911. [CrossRef]
17. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed.; O'Reilly Media: Sebastopol, CA, USA, 2019.
18. 3DBenchy. 3DBenchy Brochure. 2015. Available online: http://www.3dbenchy.com/wp-content/uploads/2015/04/3DBenchy_Broschure_3DBenchy.com_.pdf (accessed on 31 July 2024).
19. Wang, Y.; Tang, M.; Zhou, Z.; Wang, C. Study on the temperature field distribution of the interface and process parameter optimization for material extrusion of PLA. *Polym. Eng. Sci.* **2024**, *64*, 718–732. [CrossRef]
20. Artillery. (n.d.). Manual de la Impresora 3D Sidewinder X1, Hornet V4. Manuals Plus. Available online: https://manuals.plus/es/artillery-hornet/sidewinder-x1-3d-printer-v4-manual (accessed on 31 July 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.