POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Precision enhancement of the 3D printing process using Automated visual error detection

Laurea Magistrale in Mechanical Engineering - Ingegneria Meccanica

Author: Luca Alietti

Advisor: Prof. Dr. Hamid Reza Karimi

Co-advisor: Dr.-Ing. Michael Lütjen

Academic year: 2021-2022

1. Introduction

This study focuses on the post-process quality assessment for the precision enhancement of the 3D printing process. It is carried out by developing a Python code that is capable of automatically identifying errors between the printed model and the designed one and linking back them to the corresponding G-Code lines, providing the user with a tool to understand which action should be taken in order to compensate for them. In the analysis, two different data-sets have been used: a part of a shoe sole, printed using the Fused Deposition Modeling technique, and a simple component with a smooth cylindrical shape, used only to verify that the algorithm works properly.

2. Methodology

The two models are provided as point clouds, i.e. set of 3D points that describe and define the external surface of a model; the Reference (or Target) represents the designed model, while the Source is the cloud obtained from the printed component. They are firstly aligned with a 3D registration algorithm, the Iterative Closest Point algorithm [1]: it allows to progressively project the Source cloud into the same coordinate system as the Target, in such a way to have a suitable condition to compare them and define deviations in the next steps.

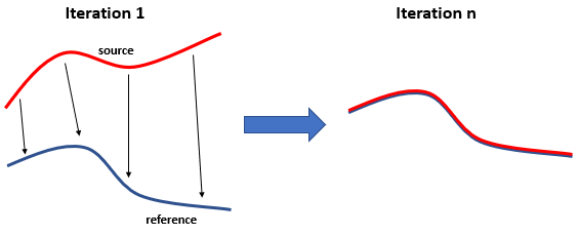

Figure 1: ICP behavior.

The algorithm can be described as a least-square minimization approach, with the following objective function to be minimized:

$$d(R,t) = \frac{1}{N_x}\sum_{i=1}^{N_x} ||x_i - (Rp_i + t)||^2 \qquad (1)$$

In eq. 1, R and t are, respectively, the Rotation matrix and the translation vector that are iteratively improved by the algorithm to obtain the optimal alignment of the two clouds; x_i and p_i are the i-esimal reference point of the Target cloud and its closest correspondence Source

point. The algorithm progressively improve R and t to reduce the distances between the points of the clouds and minimize the objective function, registering the Source over the Reference.
After the registration, deviations between the clouds are defined and the errors are detected. This is achieved by evaluating the distances between the points of the two clouds and comparing them to a threshold, which allows to identify the greater as error points. To determine the distance values a Nearest Neighbour Search algorithm has been implemented, together with the construction of a kD-Tree data structure over the Source point cloud; these were necessary because there was no possibility to define by default the right correspondences between the points of the two clouds. To identify the neighbor points means to define, for each point x of the Target, the point p^* of the Source cloud such that [4]:

$$p^* = arg \min_{p \in S} dist(x, p) \tag{2}$$

Where $dist(x, p)$ denotes the distance function between the query point x and the i-esimal point p belonging to the Source point cloud; there are different metrics to define it, the one commonly used for the definition of neighbours is the Euclidean distance.
However, the clouds have usually large sizes - i.e. they are composed of a huge amount of points - and the standard approach to define the neighbors is not a viable solution, since it requires long computation time. kD-Tree has been used to overcome such a problem, because it allows to reduce the amount of points that must be analyzed to define the neighbours, limiting them to only those are close to the Target query point. This allows, in turns, to strongly reduce the time consumption - which can be approximated to $O(n * log(m))$ [2]. The result is the partition of the Source data by several sub-boxes, each of them containing one point, that can be quickly analyzed to define the corresponding neighbour points and their distances from the Target points. These distances are then compared to a threshold, defined directly by the user in accordance with his requirements and needs; the points characterized by a distance value higher than the considered threshold are detected as error points and stored in an array. Depending on the value of the threshold, different results are obtained: with small threshold it is possible to detect even micro-deviations - increasing the number of the detected errors, while using a large threshold only macro-errors are detected.
Finally, the error points are linked back to the G-Code lines responsible of their creation; this allows the user to understand where errors come from and, eventually, to know where he has to act to compensate for or limit them. The important information that the user should get from this step are fundamentally two: the raw line of the G-Code instruction and the respective line number; in particular, the latter is the most important because it allows to precisely know where the line is positioned in the G-Code file, saving the consumer a lot of time when searching for it. These two information are provided in a text file given as output by the code, so that the user can easily visualize the "error lines". An example is reported in figure below:

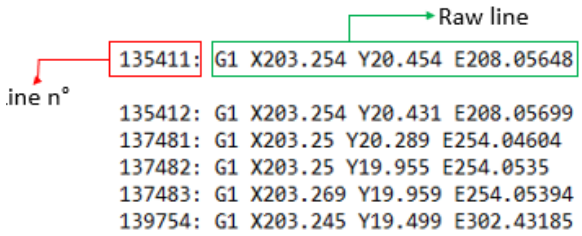

Figure 2: Output structure.

2.1. Python libraries

Within the Python code different libraries have been used; the most important are:

- NumPy: used for handling arrays and matrices;
- Open3d: used for clouds representation and ICP algorithm implementation;
- SciPy: used for the implementation of the kD-Tree and the Nearest Neighbor Search algorithm;
- Matplotlib: useful to obtain plots and images for some output representations.

3. Primary data-set

The technique has been validated with a FDM 3D printed sole of a shoe built with Polylactic Acid (PLA), a particular material able to guarantee plasticity and toughness of the component in the long-time service [3]. The Reference point

cloud is obtained by analyzing the G-Code file line by line and considering only those points that belong to the surface of the designed model; these are stored in an array and then converted in a .ply file to obtain the corresponding point cloud. The Source, instead, is obtained by scanning the printed component after the completion of the printing process.

(a) Target point cloud (b) Source point cloud

Figure 3: Input point clouds.

As explained in sec. 2, the clouds are firstly registered to put them in the same coordinate system. To provide a good result at the end, it was important to obtain a suitable alignment; this is guaranteed by an index - the fitness value, which is an indicator of the overall alignment of the two clouds: the closer it is to 1, the better is the registration. In this case it is equal to $9.9902e{-}01$ and the clouds are well aligned, as can also be seen from the registration result in fig. 4 below:

Figure 4: Registration result.

After the registration, those points that are characterized by deviations higher than the set threshold are detected as error points; the algorithm stores them in an array and produce a visualization of them over the Target point cloud. In that way, exploiting the connection between the Target and the G-Code set of instruction, it was then possible to link the errors to the corresponding G-Code lines and obtain the output file with the error lines information shown in fig. 2. An example of the visualization of the error points is reported here:

Figure 5: Errors.

Actually, the results are strongly influenced by the threshold value. This constitutes the minimum value of the distance between a target point and its defined neighbour that has to be detected as error: when the distance value is greater, an error is detected and the corresponding point is identified and stored in the array. This value has to be chosen directly by the user before running the code and it depends on his requirements and needs: if he wants to detect short deviations, the threshold needs to be small in such a way to allow the algorithm to find even those distance values that are unnoticeable; if, instead, the user is more concerned about the macro-errors - i.e. those points that are strongly deviated from the reference model, the threshold can be set higher in order to detect only huge deviations. This decision influences the number of points detected as error and, consequently, the size of the array containing these points; de-

pending on that, the generation of the output file requires more or less time. Hence, the user must handle this according to his necessities: detecting small deviations means to reach a good accuracy, but on the other hand the time consumption could be long.

4. Secondary data-set

The secondary data-set has been used to verify that the developed algorithm works properly. It represents a cylindrical shape components where, in the printed component point cloud, some points are shifted from their initial position; in that way, some deviations have been created between the scan and the reference model, in such a way to know where errors should be detected by the algorithm. The total number of modified points is 931, which corresponds to 7 points for 133 successive layers; setting the threshold value lower enough to detect all the deviations, the result shows that the only detected error points are those that have been modified in the scan point cloud (fig. 6), as expected.

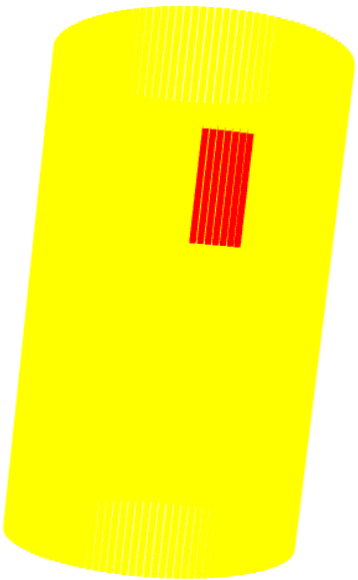

Figure 6: Errors.

This verification allows also to ensure the scalability of the methodology: the algorithm can, indeed, be used with any type of data-set, as long as the inputs are provided in the form of point clouds.

5. Conclusions

In this work, an algorithm for the self-detection of errors after the 3D printing process has been developed. This allows to automatically detect deviations between the model of the printed object and the reference model and connect them to the G-Code lines responsible of their creation, generating an output file that provides the user with a tool that allows to easily understand the G-Code problem lines. The two models are provided in the form of point clouds, describing the surface of the parts; they are firstly registered and then the point-to-point distances are defined. These are successively compared to a threshold in order to identify larger deviations and detect them as errors. Finally, the G-Code lines responsible of the generation of these errors are visualized, providing the user with an easy tool to find them within the G-Code file itself.
The methodology shows a good accuracy - depending on the value of the threshold given by the user, and it can be scaled for different data-sets; the unique constraints that has to be respected is the format of the inputs file: to be able to use the algorithm, in fact, the input file must be in the form of point clouds. The only problem that can reduce the use of this algorithm is related to the size of the clouds; for large dimensions point clouds - i.e. clouds composed of huge amounts of points, the technique needs long computation times, especially for the generation of the output file for the visualization of the G-Code lines. Anyway, the clouds of the primary data-set are composed of almost 150.000 and 500.000 points and show time problems only when small value of the threshold are given. In this case, it is up to the user to decide whether to change the threshold a little - to reduce the calculation time - or to keep this value and ensure more accurate error detection.

References

[1] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[2] Jan Elseberg, Stéphane Magnenat, Roland Siegwart, and Andreas Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registra-

tion. *Journal of Software Engineering for Robotics*, 3(1):2–12, 2012.

[3] Zengguang Liu, Yanqing Wang, Beicheng Wu, Chunzhi Cui, Yu Guo, and Cheng Yan. A critical review of fused deposition modeling 3d printing technology in manufacturing polylactic acid parts. *The International Journal of Advanced Manufacturing Technology*, 102(9):2877–2889, 2019.

[4] Parikshit Ram and Kaushik Sinha. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, pages 1378–1388, 2019.