Title:
Automated Reconstruction of Continuous Robotic Motion from G-Code Patterns

Authors:
Wout De Backer, wdbacker@email.sc.edu, McNair Center, University of South Carolina
Max B. Kirkpatrick, mbk@email.sc.edu, McNair Center, University of South Carolina
Ramy Harik[1], harik@mailbox.sc.edu, McNair Center, University of South Carolina
Joshua A. Tarbutton, jat@sc.edu, McNair Center, University of South Carolina

Keywords:
Feature Recognition, Process Planning, Numerical Control, feature extraction, Toolpath Optimization

DOI: 10.14733/cadconfP.2016.xxx-yyy

Introduction:
Links between Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) packages have been established through translators and dedicated tools for decades. Tools such as CATIA © or Mastercam © allow users to generate toolpaths for a large variety of machines, while maintaining peculiar machine specifications. Current developments in open source manufacturing, driven largely by the 3d printing hobbyist community, are resulting in a booming of the toolpathing technology tools and applications for additive manufacturing.

The emergence of open source tools for additive manufacturing, which often discretize continuous motion in finite linear segments, offer an interesting opportunity: these tools are optimized to fill or trace a defined surface or slice of which the bound is defined from an intersection with a simplified surface tessellation model. The latter contains a discretized solution of a parametric feature as defined by a CAD program. Therefore, these tools are ignorant of any potential parametric surfaces or curves present in the topology to be traced. For many, more industrial, machines, the discretized line segments which define an arc, spline or other feature are not optimized and neglect higher order functionality of the ability of the machine to be programmed.

Current industrial manufacturing tools are capable of processing an impressing flow of data and their ever improving accuracies assure that nearly any shape can be (re)produced accurately. Although, as long as tessellation is used, a significant amount of data is required to represent any arc which could be easily replaced with a single arc command from the manufacturing tool. Many current industrial manufacturing tools contain some functionality to execute arc or spline movement, which is beyond what the linear tessellation is able to indicate. The detection and replacement of current code with these higher order toolpathing functions leads to a more accurate, more efficient toolpathing code which simplifies not only the simulation process, but can make it easier for the operator and machine to interpret and simplify the massive amount of coordinates that would otherwise be required to define a certain path.

This abstract aims to prove a methodology for rediscovering the parametric, higher-order features using the patterns present in the non-parametric model. A proof of concept to demonstrate the re-establishment of the parametric toolpath based on a non-parametric G-Code file is applied to a KUKA robotic platform. Sample geometry is processed and simple G-code toolpathing commands are generated with open-source developed software, containing only linear movement commands and no higher order functionality of a more developed system. This simple toolpath generation is then cleaned up and scanned for patterns and features, which can then replace sections of code in higher

[1] Corresponding author: harik@cec.sc.edu

order functional toolpathing code. For the KUKA Robotic platform, Kuka Robotic Language (KRL) output was generated containing circular arc commands, and for-loops.

State Of The Art:
Recreation of optimal toolpath from discretized data is well researched in the literature [03-04-13]. This is particularly of interest to properly analyze and represent additive manufacturing parts [6]. It falls under the general context of support for computer aided process planning (CAPP) applications [11-12] leading to appropriate analysis for potential machining features [14]. A methodology for reconstructing an unknown surface from a set of scattered points is described in [02-08]. The ability to have boundary with complicated topology, and the conversion to splines ensured a successful attempt. The main setback was the non-parametrization of the construction, making the obtained model unusable in subsequent steps. A review on approaches for handling usable curves from CAD to CAM, in the Bezier format, is conducted in [1]. The authors identify multiple toolpath planning that lead to machining time reduction which can be appropriated. Recently, graph theories [09] and approximation methods [05], as well as many other topological operations, are investigated and successful specific results are obtained. [05] is particularly interesting to our scenario where circular arcs were a starting point for this research. Finally, even though a lot of this research was initiated with cutter locations in subtractive manufacturing [07-10-15], multiple benefits can be identified and applied for additive manufacturing.

Methodology:
G-code output from open source developed toolpath generators such as Slic3r is intrinsically limited, and produces parts that approximate the actual geometry. Raw G-code used in 3D printing is prone to infinitesimal errors stemming from the tessellation of the parametric surface. This often produces an inefficient and less optimal toolpath for the reproduction of the geometry. In order to create a better representation of the actual geometry, a higher order language is therefore desired to describe geometries and movements in a way that many open-source slicer-generated G-code cannot. Each line of the input G-code is read and stored into a list of relative movements for the machine to follow. For most applications, this list contains the relative X,Y,Z, E (Extrusion) and F (Feedrate) commands for every point the end effector must travel to. If higher order G-code was input, this list also contains the center point of any arc commands and the arc's direction (clockwise or counter-clockwise), as defined by G2 and G3 commands. Every command from the original G-code is in this list, simply converted from absolute coordinates to relative coordinates. This distinction is important, as the change from absolute to relative coordinates allows the code to be generalized to any system, regardless of the location of the system's origin. This master list of all the commands from the original G-code is then parsed into discrete layers or passes of the end effector by monitoring the z-coordinate of every point in the list. These layers are then passed off to a number of different optimization algorithms, described in the subsections below.

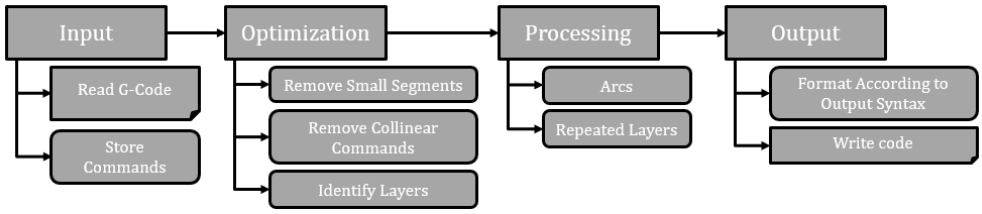


Fig. 1: The feature recognition process for updating lower code into higher-order machine code.

Collinearity Recognition
The output G-code from many slicers contains a number of unnecessary moves or moves that could be concatenated into one command in the same direction, often originating from the tessellation of the geometry file. The removal and concatenation of these commands eliminates unnecessary complexity in the toolpath, which is essential for later methodologies that are to be applied to the master list. Collinear commands are found by examining the orientation of each movement vector, and through comparison of that vector to the previous vector. If the two movements are aligned within tolerance,

the second command is removed, and the first is replaced with a single vector that is the sum of the original two vectors, as shown in figure 2. This analysis is applied to every sequential pair of points for every layer in the part. After this process is concluded, the master list now contains a set of commands without redundant collinear commands.

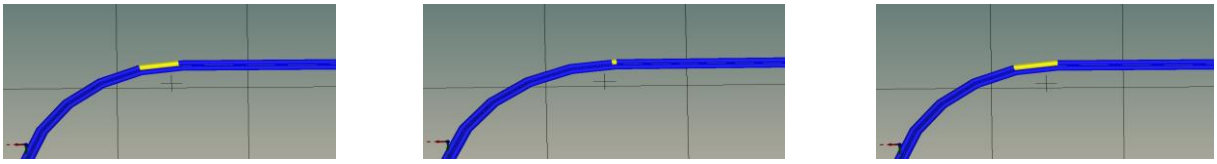


Fig. 2: Collinear movements that can be concatenated into a single command: (a) The first command, (b) The second command, and (c) The concatenation of the two commands into one command.

Further Toolpath Cleanup
In addition to removal of collinear commands, other small and insignificant moves may be eliminated from the toolpath in order to create a cleaner path. The insignificant moves removed are all moves with lengths that are under set machine tolerances and accuracies, as shown in figure 3. Commands above the tolerances are modified such that the insignificant moves are incorporated in the final toolpath.

Arc Feature Recognition
The optimized master list can then be parsed into layers to perform topological feature recognition analysis. The parser then tries to regenerate the original parametric feature based on the geometry from the linearized G-code. For arc commands, the parser must decide what commands may be considered as part of an arc, by comparing the direction change between any two sequential commands and comparing this to the previous direction change. If a set of movements all have direction changes within tolerance and the lengths are equal within tolerance, then that entire set may be part of an arc. One must note that identifying if a set of points is part of an arc is not trivial and sometimes this concatenation is not desired, so the tool must be used with caution. Figure 3c shows a set of moves that are considered as part of an arc by the parser.

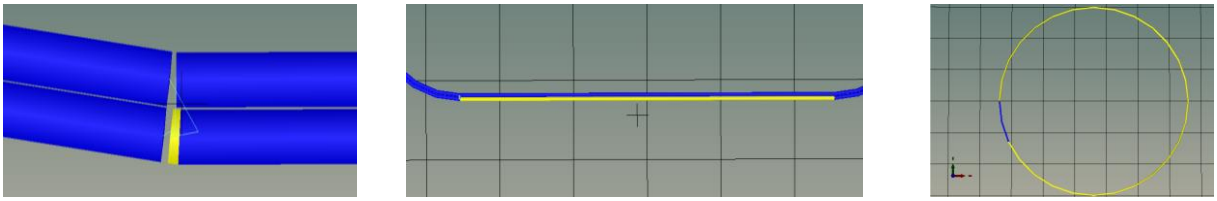


Fig. 3: Movement commands and their resulting concatenation: (a) An unnecessary short movement command, (b) The concatenation of that movement command with the long movement command trailing it and (c) A set of concentric moves that can be converted into a single arc movement.

Repeated Layer Recognition
In addition to recognition of topography in individual layers, the parser also seeks to find any layers that are repeated and use for-loops in a higher level language to repeat them. The parser accomplishes this by comparing the coordinates for every point (including arc commands) in the current layer to the coordinates of every point in the previous layer. This allows parts with consistent cross sections to be printed in very few commands (as little as 10), where before the slicer output may have hundreds of thousands of points. Repeated layer recognition along with 2d arc recognition allows for massive reductions in output file sizes, as well as a more faithful recreation of the original geometry.

Code generalization & Output to machine language
The master list is generalized to a format that can be converted into any higher level language, involving the making of a set of all repeated layers, arcs, and any additional points that may aid any conversion, such as arc centers. This list can then be parsed into another higher level machine language. This particular parser was written to output KUKA Robotics Language (KRL), however any machining language could be used for output, as long as the syntax is understood by the programmer. The master list is written out entry by entry into commands in the higher level language, and the additional set lists are used to determine where for-loops and arc commands may be placed, if applicable in the chosen higher language. The output language may even be G-code, and if the input language was linear G-code, then linear and arc-supported G-code may be output to increase code efficiency.

As shown in figure 4, for the case of a stadium-shaped extrusion, the ideal parser would be able to convert the complex toolpath with hundreds of points and layers into a 6-point, for-loop controlled simplified toolpath, significantly reducing the required data and optimizing the toolpath for a higher order code.

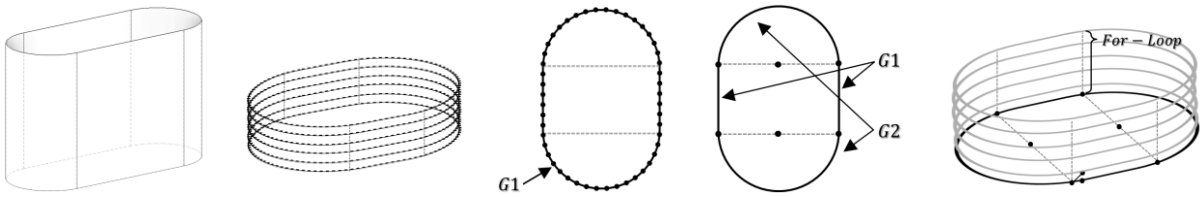


Fig. 4: The optimizing of a stadium toolpath: (a) The desired geometry, (b) An example of an original toolpath, (c) One layer from the original G-Code with sufficient linear movements to approximate the curves (d) One layer of the optimized path with 2 linear commands and 2 arc commands (e) Representation of the optimized toolpath code including a for-loop.

Application:
In order to get a better representation of manufactured parts with respect to their digital counterparts, and to correct issues with the raw linear G-code, an in-house G-code parser was developed which cleans up toolpathing G-code, identifies higher-order functions and compiles these functions in a higher order robot programming language to form a more optimal, memory-efficient machine-independent code, according to the methodology described above. The parser then outputs the code according to an output function which is determined by the syntax of the application that is desired. The general scope of the application is shown in figure 5.

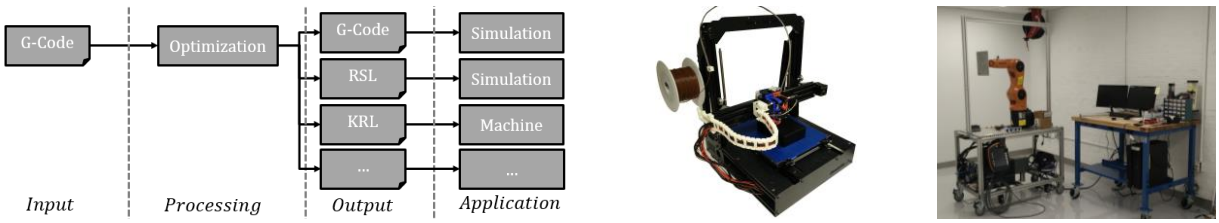


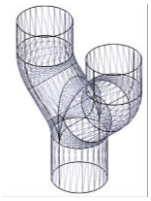
Fig. 5: From left to right: (a) Example of the scope of the post-processing workflow with the in-house developed G-code optimizer, (b) The Mendel Max 3, an off-the-shelf 3d printer controlled by G-code and (c) An in-house developed 3D Printer based on a KUKA Robotic platform used to run the converted code, controlled by KRL-code.

KRL Application
The G-code modification process was applied to an industrial KUKA Robotic Platform. This platform was converted into a 3D printer by reading and converting open-source generated G-code and into Kuka Robotic Language toolpathing commands which fit the robot controller. With the tolerances of

the system in mind, the insignificant length is set for 0.2mm long. This value is low enough that the removal and concatenation of these commands will have a negligible effect on the output geometry, as 0.2 mm is lower than the nozzle diameter of the printer. The controller allows circular commands as well as other higher-order functionality such as for-loops.

Examples of Toolpath Processing

Three examples were chosen to demonstrate the toolpath processing: a simple cylinder, an elongated stadium, as shown in figure 4, and a more complex Y-shaped geometry, as shown in figure 6a. These were sliced using Slic3r, which uses the tessellated CAD surface as base for the toolpath commands, and run through the optimizing parser. The results of the optimization can be seen in figure 6b.



	Original	*Optimized*	*Reduction*
Cylinder	13801	11	99.9%
Stadium	44838	259	99.4%
Y-Shape	44918	22379	50.2%

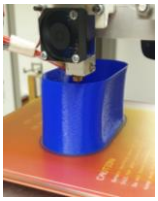


Fig. 6: Examples and results of the optimization: (a) The Y-geometry example, (b) The effect of the optimization on the number of commands for a simple and complex geometry, (c & d) The stadium geometry 3D printed on the KUKA setup proving the successful translation of G-Code to optimized KRL.

Conclusion

The need for a reverse topological toolpath feature recognition process is identified and, based on an industrial application, a parser was developed while keeping the allowing for generalization of the process. The G-code of sample artifacts ranging in complexity was processed by the in-house developed parser, analyzed and processed successfully, reducing the output commands while increasing the toolpath efficiency and accuracy. The developed tool was able to successfully omit infinitesimally small and redundant features, detect repeated features and modify the toolpath in such a way that the resulting part forms the same or a better approximation of the desired geometry, for a range of complexity of the base code, paving the way for future efficiency and topological toolpath feature improvements.

References:

[1] Fitter, H. N.; Pandey, A. B.; Patel, D. D.; Mistry, J. M.; A review on approaches for handling Bezier curves in CAD for Manufacturing, Procedia Engineering, 97, 2014, 1155-1166, http://dx.doi.org/10.1016/j.proeng.2014.12.394

[2] Guo B.: Surface reconstruction: from points to splines, Computer-Aided Design, Vol. 29, No. 4, 1997, 269-277, http://dx.doi.org/10.1016/S0010-4485(96)00055-3

[3] Kim D-S.; Jun C-S.; Park S.: Tool Path Generation for clean-up Machining by Curve-based Approach, Computer-Aided Design, 37, 2005, 967-973, http://dx.doi.org/10.1016/j.cad.2004.09.023

[4] Lin C. A.; Liu H-T.: Automatic Generation of NC cutter path from massive data points, Computer-Aided Design, Vol. 30, No. 1, 1998, 77-90, http://dx.doi.org/10.1016/S0010-4485(97)00066-3

[5] Liu, Z.; Tan J.; Chen X.; Zhang L.: An Approximation method to Circular Arcs, Applied Mathematics and Computation, 219, 2012, 1306-1311, http://dx.doi.org/10.1016/j.amc.2012.07.038

[6] Nelaturi, S.; Shapiro, V.: Representation and Analysis of additively manufactured parts, Computer-Aided Design, 67-68, 2015, 13-23, http://dx.doi.org/10.1016/j.cad.2015.03.007

[7] Park, S. C.; Chang, M.: Tool Path Generation for a Surface Model with defects, Computers in Industry, 61, 2010, 75-82, http://dx.doi.org/10.1016/j.compind.2009.07.003

[8] Park, S. C.; Chung, Y. C.: Tool-path generation from Measured Data, Computer-Aided Design, 35, 2003, 467-475, http://dx.doi.org/10.1016/S0010-4485(02)00070-2

[9] Popescu, D.; Popister, F.; Popescu, S.; Neamtu, C.; Gurzau, M.: Direct toolpath generation based on graph theory for milling roughing, Procedia CIRP, 25, 2014, 75 – 80, http://dx.doi.org/10.1016/j.procir.2014.10.013

[10] Ren, Y.; Yau, H. T.; Lee Y-S.: Clean-up tool path generation by contraction tool method for machining complex polyhedral models, Computers in Industry, 54, 2004, 17-33, http://dx.doi.org/10.1016/j.compind.2003.09.003
[11] Subrahmanyam S.; Wozny M.: An overview of automatic feature recognition techniques for computer-aided process planning, Computers in Industry, 26, 1995, 1-21, http://dx.doi.org/10.1016/0166-3615(95)80003-4
[12] Tuttle R.; Little G.; Corney J.; Clark D. E. R.: Feature recognition for NC part programming, Computers in Industry, 35, 1998, 275-289, http://dx.doi.org/10.1016/S0166-3615(97)00089-4
[13] Varady T.; Ralph R. P.; Cox J.: Special Issue: Reverse engineering of geometric models, Computer-Aided Design, Vol 29, No. 4, 1997, 253-254, http://dx.doi.org/10.1016/S0010-4485(96)00053-X
[14] Wu M. C.; Liu C. R.: Analysis on machined feature recognition techniques based on B-rep, Computer-Aided Design, Vol. 28, No. 8, 1996, 603-616, http://dx.doi.org/10.1016/0010-4485(95)00075-5
[15] Wu, J.; Zhou, H.; Tang, X.; Chen, J.: Implementation of CL points processing methodology with NURBS curve fitting technique for high speed machining, Computers and Industrial Engineering, 81, 2015, 58-64, http://dx.doi.org/10.1016/j.cie.2014.12.018