UNIVERSITY OF CALIFORNIA

Los Angeles

Improvements and Characterization of Error, Speed, and Capabilities in Low-Cost Additive Manufacturing Systems

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in Mechanical Engineering

by

Hannelore A. Hemminger

2022

ABSTRACT OF THE THESIS

Improvements and Characterization of Error, Speed, and Capabilities in Low-Cost Additive Manufacturing Systems

by

Hannelore Azora Hemminger

Master of Science in Mechanical Engineering
University of California, Los Angeles, 2022
Professor Xiaoyu (Rayne) Zheng, Chair

Herein, I present advancements in path and trajectory planning in low—cost Fused Filament Fabrication (FFF) additive manufacturing. I improve and extend path and trajectory planning algorithms with a focus on low computational cost to enable real-time computation on limited hardware such as a Raspberry Pi or similar. I present a large dataset of vibrational overshoot error and simulated motion completion times classified by characteristic types of motion, to pave the way for section—specific setting optimization for the improvement of FFF 3D printing processing speeds. I demonstrate a novel approach for manufacturing autoclavable Personal Protective Equipment (PPE) using low—cost modifications to a low—cost, open—source 3D printer, potentially allowing distributed manufacturing of autoclavable PPE in times of supply chain uncertainty, such as during the COVID-19 pandemic. Finally, I present a cost—reductive approach to the motion stages and control system of a complex additive manufacturing research system.

The thesis of Hannelore Azora Hemminger is approved.

Jacob Rosen

Jonathan B. Hopkins

Xiaoyu (Rayne) Zheng, Chair

University of California, Los Angeles

2022

Table of Contents

List of Figures

List of Tables

Thesis Introduction

In industry 4.0, accelerated manufacturing speed, agility, and turn-around times are more accessible than ever before with the advent of additive manufacturing. With a broad material portfolio and the ability to manufacture previously impossible geometries, the additive manufacturing industry has proven itself as a major player in industry as well as in academia. The reduced turn-around time has enabled a faster design-to-prototype cycle than previously possible, and in the last decade has seen widespread industry adoption in engineering firms as well as the consumer market [1-3]. This has allowed an unprecedented number of engineers, entrepreneurs, inventors, and garage tinkerers access to a manufacturing technology that prior to the last decade had been too expensive for individuals and small businesses [1, 4]. This has led to the birth of accessible and reliable low-cost 3D printing via Fused Filament Fabrication (FFF) and, more recently, photopolymerization processes [5, 6]. This in turn has given rise to the need for smarter path and trajectory planning which are compatible with low-cost hardware on the end-use machines. Classically, research on path and trajectory planning in academia has focused on real-time control algorithms, especially input shaping and filtered b-spline basis function approaches [7-10].

However, this necessitates expensive hardware, especially when feedback control is incorporated into the mix. Most current papers rely on high-cost controls equipment from companies like National Instruments or dSpace [11]. This has created a divide in additive manufacturing between well-funded, formalized research in industry and academia and low-cost development done by open-source companies and communities. Several companies, including Prusa Research, E3D, Ultimaker, Creality 3D, and Lulzbot have worked to improve low-cost and open-source hardware designs in the FFF additive manufacturing field, and

sometimes photopolymerization technologies [12-16]. Similarly, open-source communities have risen around popular enthusiast 3D printer designs such as Voron, HyperCube, D Bot, Railcore, V-Core, HevORT, and others [17-22]. These printers are typically much faster, which in turn has even given rise to a small community of what can only be described as the drag racing of 3D printing – fastest completion time for a Benchy model, quality optional [23]. Unfortunately, industry and academia rarely, if ever, give the time of day to low-cost applications due to the lack of available research funding for it. In low-cost development communities, the enthusiasts and tinkerers contributing to the open-source designs typically have little available time and often have limited formal experience in engineering and research, so they may lack the technical background needed to understand and apply much of the current academic research.

These low-cost machines also form the basis for how engineering students and the public understand and use 3D printing, as advanced equipment may cost hundreds of thousands or even millions of US dollars [24]. This extends into engineering firms' prototyping equipment, where the value of putting affordable 3D printers on the desks of engineers to speed up prototype turn-around time is undeniable, relying on low-cost 3D printers turning around quality prototypes as quickly as possible [25]. Therefore, both industry and open-source 3D printing communities share a goal in additive manufacturing: speed up FFF 3D printing while maintaining or increasing part quality.

In service of this goal, it is necessary to develop an understanding of the variables that can be controlled which affect the speed and accuracy of 3D printing, and to determine the relationships between these variables and the resulting print throughput and accuracy. When focusing on FFF technologies, the controllable parameters fall under two categories – thermodynamic (limitations resulting from the liquefaction and cooling dynamics of the build

material and part) and kinematic (limitations resulting from the physical motions of the machine, arising from motor torque limitations or, more often, induced vibrations). The thermodynamic challenges have received some focus from academia of late, due to the interest in large-format additive manufacturing, for which the liquefaction dynamics of the extruder play a crucial role [26, 27]. However, the kinematic limitations are more neglected, and the little research being done in this area is typically focused on real-time control techniques run on expensive hardware [7, 9]. This leaves a gap in additive manufacturing research that must be filled make high-tech path and trajectory planners achievable on the low-cost hardware available to low-end 3D printer manufacturers and open-source communities that need it.

Herein, I focus on the kinematic side of FFF 3D printing technology and seek to determine and improve the relationships between various motion parameters and overall print speed and tracking error, reconciling advanced path and trajectory techniques with low-cost hardware limitations. In addition, I discuss the design and application of a low-cost, easily accessible solution for the additive manufacturing of autoclavable PPE, as well as considerations for cost reduction in the control of a complex novel additive manufacturing system.

Most code and data for this thesis are available on an open-source basis at the following site: https://osf.io/6x7cg/

Chapter 1 – Jerk-limited, Smart-cornering Path and Trajectory Planner for FFF Additive Manufacturing Systems

Introduction

In modern engineering and industry 4.0, additive manufacturing is being leveraged at every level of the design stage from prototyping to low-volume production to bring ideas to physical reality faster than was ever possible with traditional manufacturing [28-30]. Critical to this is the ability of additive manufacturing machines to produce parts as quickly as possible while staying with an acceptable error tolerance on the parts produced. While a wide variety of AM technologies have risen up over the last decade, when it comes to fast turnaround prototypes in engineering design, FFF remains the most commonly used technology due to its low cost and accessibility [24]. These characteristics have also led to an explosion in the consumer 3D printing market, with the cost of FFF (often called FDM – Fused Deposition Modelling – in consumer spaces) 3D printers plummeting to the level of obtainability for garage tinkerers, inventors, and entrepreneurs alike. It is desirable, then, to optimize the part completion times of these machines without sacrificing part quality to do so. This would allow for shorter design cycles and faster time to market. Since plastic throughput is limited by the size of the extrusion bead and the average linear speed of the print head during extrusion, speeding up a print requires either worsening the part resolution by using a larger extrusion width and height or speeding up the average motion speed without increasing errors in the motion execution.

When considering a given set of motion hardware, the overall speed and error of motion execution are determined mostly by the path and trajectory planner running on the

software and firmware controlling the machine. Since FFF machines, especially those using low-cost belted motion systems, are flexible systems, there will be oscillations induced in the motion during periods of acceleration and deceleration as the belt coupling the motor and load vibrates. These oscillations result in artifacts at sharp corners in printed parts, colloquially called 'ghosting' or 'ringing' by the open-source community [31]. In addition to vibrational error, the path and trajectory planning also affect the overall rate at which the machine finishes the part. The goal of the path and trajectory planner, therefore, is to minimize these unwanted vibrational errors in the motion while simultaneously minimizing the completion time of the motion. Efforts to this effect can be broadly categorized into algorithms which operate on the path in cartesian space that the machine is commanded to follow, and the trajectory (sometimes called the time law) of the print head through time. Since the path and trajectory can be manipulated by a variety of approaches and variables, any research into the coupled effects of these on the motion quality and completion time must by necessity start with a path and trajectory planner that can control enough path and trajectory parameters to satisfactorily manipulate as many of these effects as possible. At the same time, to be applicable to the low-cost 3D printers that this work focuses on, the path and trajectory planner must not require any computationally expensive real-time control algorithms or expensive real-time feedback hardware. This chapter seeks to create a such comprehensive path and trajectory planner and implement it on low-cost hardware, implementing and improving upon the path and trajectory planning algorithms in present literature, all the way from the parsing of the machine G-Code to the physical motion of the stepper motors. Implementation is done in Matlab, with the code files posted open-sourced and the key logic of each portion explained and discussed so that the method can be easily

implemented by future researchers and open-source communities in any preferred programming language.

Background and Literature Review

Path planning, in general, focuses on how best to interpolate a path through a series of pre-determined waypoints such that undesired vibration in the motion is minimized while the completion speed of the motion is as fast as possible. Presently, the most advanced open-source FFF 3D printer firmware, Klipper, applies circular arcs at each junction so that the machine does not have to turn a perfectly sharp corner, allowing for higher average motion speeds [32]. However, this method is not without issues – in particular, it is only G1 continuous due to the junction between the straight-line elements and the arc elements inducing an instantaneous change in curvature in the path. This results in non-ideal trade-offs between motion speed and vibration error, as evidenced by experiences reported by users of the firmware [33]. Specifically, this curvature discontinuity results in an effective discontinuous acceleration applied to each motion axis trajectory, even when the corner is taken at a constant speed. Therefore, while the kinematics for this solution are simple to compute on limited hardware even in real time, it results in induced vibrations in the print results for even modest cornering speeds. Therefore, a more advanced cornering algorithm is needed.

On the other hand, in industry and academia, a variety of more advanced (but computationally more expensive) cornering algorithms exist. Most cornering algorithms in current literature revolve around various forms of Bezier splines, which are useful due to the ability to easily guarantee G2 continuity [34-37]. Sencer et. al. proposed a Bezier spline corner blending algorithm which uses six control points, chosen such that three control points

each lie along the lines between the previous and next waypoint, equally spaced, with their locations along these lines determined such that the curvature is optimized at the corner even for varying cornering angles. The speed at which these corners are taken is found iteratively to guarantee kinematic compatibility (so that there will always be enough time for the machine to accelerate or decelerate in time for the next corner's cornering speed). Zimenko et. al. simplifies the computation of this method, although in doing so discard the curvature-optimal quality of the algorithm. Neither Sencer's nor Zimenko's trajectory planning algorithms make use of a time-optimal federate profile during the corner, and instead simply use convenient formulations for maintaining trajectory tangentiality. Since this can affect both the federate within the corner as well as the achievable feed rates in the neighboring moves, this is an aspect of the 6-point Bezier spline corner smoothing algorithm which can be improved upon at little computational cost by applying jerk-limited trajectory generation algorithms from elsewhere in the field. In addition to this algorithm, other techniques exist using Bezier splines. Frequently, these techniques are similar to the one discussed above but using a slightly different choice of control points or order of the Bezier spline, such as in Ozcan et. al. [36]. Notably however, recently a more novel Bezier spline cornering algorithm was proposed by Huang et. al. [37]. In their proposed algorithm, the toolpath is permitted to 'swing out' within the tolerance bounds in a convex manner leading up to each corner so that it can cut across it at a reduced angle, thereby further reducing the maximum curvature of the toolpath. However, this does come at the cost of increased computational complexity as well as less predictable error shapes, which could make it more complicated to measure vibration error in the resulting trajectory in an automated manner. A table summarizing the most important aspects of each of these approaches compared to the approach taken in this work is shown below in Table 1:

Table 1: Comparison to other algorithms

Algorithm	G2 Continuity	Curvature Optimization	Jerk-Limited	Intra-corner Acceleration	Entire Trajectory Considered	Simple Computation	Simple Error Shape
This work							
Sencer							
Zimenko							
Ozcan							
Huang							

In addition to Bezier spline-based path planning techniques and analytical and numerical trajectory generation approaches, in recent years the advent of machine learning has led to a new sub-field in path and trajectory planning in which machine learning is used to determine the path and trajectory (sometimes simultaneously) of the machine tool [38-44]. Deep learning and deep reinforcement learning models have dominated this subfield, yielding promising early results. Applications of this method are not limited to machine tools – in fact, machine learning has been applied to trajectory generations even in the field of air traffic control algorithms in Guan et. al. [38]. Even among more classical machine control applications, these machine learning methods approach the problem of path and trajectory generation from a variety of angles and for a variety of systems. Zhou et. al. apply machine learning to train an inverse kinematics model by generating learning-ideal spline-based trajectories to generate the dataset used for the machine learning model in a structured way to out-perform more intuitive trial-and-error approaches to dataset generation used by previous machine learning-based algorithms [41]. However, these approaches are typically

real-time control-based approaches which operate on closed-loop systems at high computational cost.

In parallel and interrelated to path planning, the field of trajectory generation arose from the requirement to limit motion characteristics to physically manageable levels for the motors. This led to the use of piecewise polynomial functions to form time-optimal trajectories according to the limits set, with the order and piece count of these piecewise functions varying based on the number of time derivatives of the trajectory which are desired to be controlled as well as the required continuity characteristics of the resulting trajectory. Initially, this began as second-order polynomial function segments, also known as acceleration-limited or trapezoidal-velocity trajectories due to the highest derivative limited being acceleration and the shape of the resulting velocity profile being that of a trapezoid, as can be seen in Figure 1. However, higher order basis functions have been shown to be desirable for reducing vibrations as well as better regulating actuator force and torque output [45-47]. This has led to research on algorithms using third-order polynomial segments, also called jerk-limited or s-curve motion, and even for higher order functions which limit the snap, crackle, pop, etc. of the resulting trajectory [48]. However, as the order of the function segments increases, the number of phases also increases exponentially – and in the case of generalized boundary conditions, each of these phases of motion independently may or may not exist, making it extremely difficult to solve for higher order trajectories in the generalized case [49]. In fact, to date there has been no analytical solution found in the literature for trajectories of higher order than jerk-limited motion. Instead, higher-order profiles rely on the use of numerical techniques. One notable approach to such trajectories involves the use of averaging filters. Besset et. al. shows an approach for generated jerk-controlled trajectories using FIR filters to process acceleration-limited trajectories into jerk-limited trajectories, and later extended

this to apply to more general cases as well as higher order trajectories by cascading FIR averaging filters with procedurally calculated coefficients [48, 50]. Biagiotti et. al. extend this FIR filtering approach to cases with both kinematic and frequency constraints [51], while Xia et. al. present an algorithm that allows for changes to be efficiently made to the trajectory on-the-fly for cases in which the trajectory may need to be adjusted, such as when a human steps into the range of an industrial robot [52].

There are also other paradigms for trajectory basis functions, such as sinusoidal-square acceleration profiles [53] and spline-based trajectory functions [54]. Sinusoidal-square acceleration has the benefit of being fully continuous for all derivatives, but spline-based trajectory functions have the advantage of easily guaranteeing any desired degree of continuity while being well understood both computationally and behaviorally with respect to filtering and real-time control algorithms. However, they lack the critical characteristic of piecewise polynomial functions which makes them so attractive, which is the ability to guarantee a time-optimal solution for a set of given kinematic constraints. This makes these alternative approaches less attractive for applications in which the completion time of motion is key to the performance of the trajectory planner.

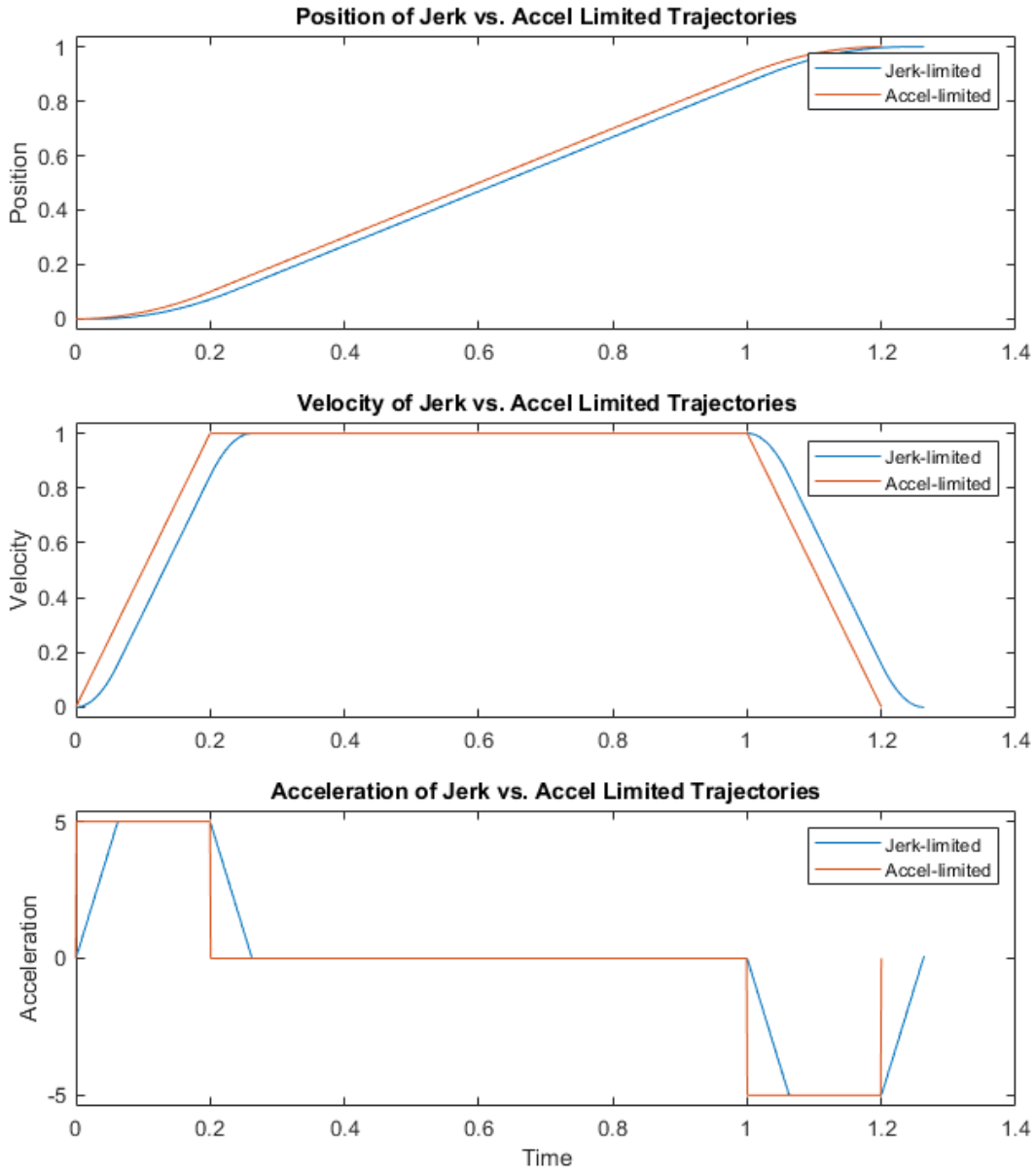

Figure 1: Acceleration-limited vs. jerk-limited trajectory illustration

Algorithm Flow and Call Stack

An algorithm flow chart is shown below in Figure 2. Note that although it is shown in the chart for illustrative purposes, G-code generation occurs within the scope of the slicing

software used rather than inside the path and trajectory planner. A detailed call stack showing the full flow of the algorithm computing and executing a trajectory from a g-code file is also shown below in Figure 3. motionTestRun is the main parent class, and the path and trajectory planner are called together in a combined member function makeTrajectory. It also contains member functions createBinary(), which prepares the binary file used to communicate the pulses to the motor drivers, and executeMotion(), which sends this binary file to the microcontroller which commands the motor drivers.

Figure 2: Basic algorithm flow

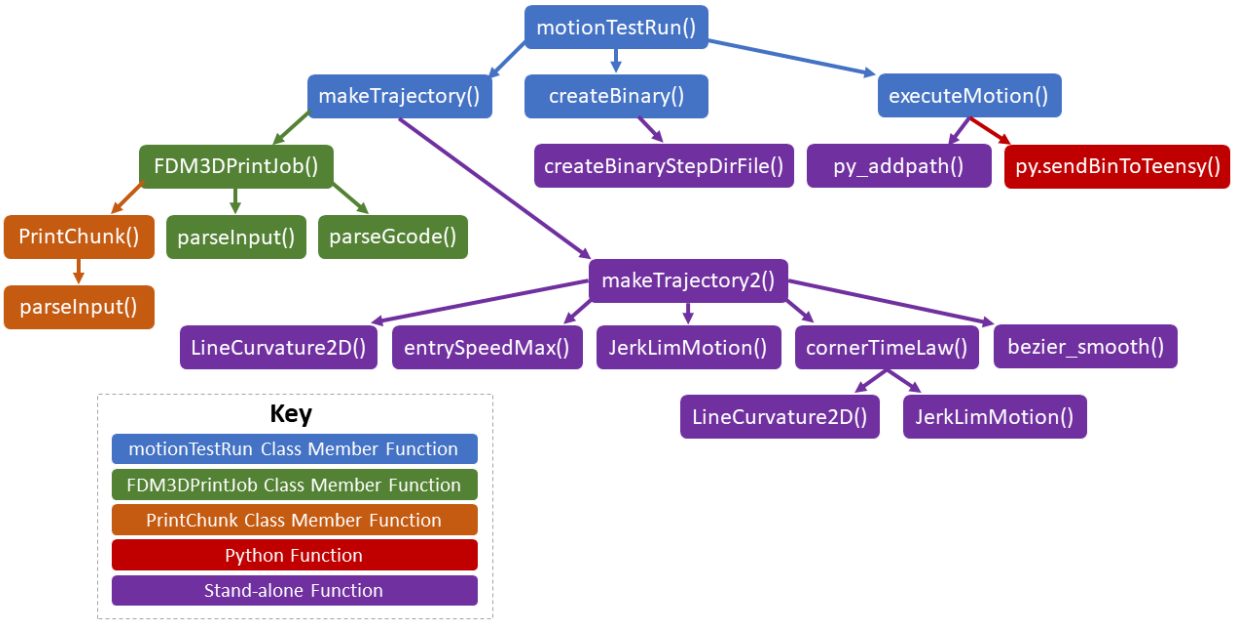

Figure 3: Main call stack of the proposed algorithm

G-code Parser

Before the path and trajectory planning can occur, the waypoints must be parsed from the g-code file. This occurs when the makeTrajectory() function is called from the motionTestRun class. This initiates a sub-class FDM3DPrintJob(), which serves as a container class for the overall print job. This sub-class then calls its member function parseGcode(), which parses the g-code file into waypoints and groups them into pieces based on the type of motion. In the present work, the determining factor used to classify the pieces is whether the extruder axis is active and whether the Z-axis is active. In this way, each individual extrudate is taken as a chunk, which results in the velocity at the beginning and end of each extrude being zero. Each of these chunks is initialized as a sub-class PrintChunk(), which serves as a container class. The major g-code commands to be considered for the proper determination of the waypoints are the G0, G1, and G92 commands, all of which affect either the commanded position directly or affect the interpretation of the position commands. Since this work is focused on path and trajectory planning, other g-code commands are not implemented, as these three commands are sufficient for the analysis of most trajectories. Once the g-code is parsed into chunks, the path and trajectory planning begin by calling the stand-alone function makeTrajectory2() on the waypoint lists of each chunk individually to compute trajectories for each chunk individually. The compartmentalization of the print job is beneficial not only to ensure that each extrudate begins and ends at zero velocity, but also to provide a convenient way for the print job to be computed a piece at a time, for example if it is desired to use a just-in-time computation paradigm for the print job to reduce memory and storage usage. Optionally, the waypoint list can also be decimated by any desired path decimation algorithm to limit the resolution of the waypoint list to, for example, reduce the number of overly short motion segments.

Path Planner

Once makeTrajectory2() is called, it begins by using a corner smoothing algorithm to smooth out corners to reduce vibration and improve overall motion speed. The selection of which corner smoothing algorithm to implement in this work was done with consideration to simultaneous curvature optimization as well as computation simplicity and predictability of the smoothed path. For this reason, the Bezier spline-based smoothing algorithm proposed by Sencer et. al. was chosen to be implemented, due to its low computational complexity while still optimizing the curvature for the Bezier spline basis used [34]. While the method proposed by Huang et. al. would have superior curvature optimization, it would also have less predictable pathing near corners due to its allowance of convex pathing around corners, as well as increased computational complexity which would make it more difficult to implement in hardware-limited applications [37]. While Zimenko et. al. marginally simplifies the computation, their proposed algorithm does not choose the spline in a curvature-optimal way, which will hurt overall motion speeds at only a very modest decrease to calculation time.

The path planning algorithm begins by computing the turn angle for each individual corner in the waypoint list, which is then used to compute the curvature-optimal parameters for the Bezier spline for the corner in conjunction with the maximum cornering error set according to the procedure outlined in Sencer et. al. In an improvement to their algorithm necessary for modern slicers which can output very high-resolution g-code, to ensure that there is always a straight-line segment between any two neighboring corners regardless of the distance between the waypoints, the transition length is capped at no more than one-third the distance of the smaller of the two segments forming the corner. In another improvement to the algorithm, the interpolated path is generated every 0.01 mm, with a minimum of 51 points to guarantee that the shape of the corner is appropriately represented.

This substantially reduces computational resources compared to the real-time interpolation at every time interval used by Sencer et. al [34]. The path is organized into individual moves, consisting of a straight-line segment followed by the corner between that motion segment and the next motion segment, except for the final move which has no corner portion. The maximum curvature of each corner is computed and in turn used to compute the maximum speed at the corner tip for which the path-normal acceleration limit will still be respected, for use during the trajectory planning phase. The maximum speed at which the corner can be entered while still reaching this maximum speed at the corner tip is also computed at this stage within the entrySpeedMax() function. A variety of other parameters needed for trajectory planning are similarly stored for the move, such as the extrusion amount per unit distance moved, start position, end position, and distance traveled within the cornering portion. Once these parameters have been computed and stored in a container struct, the trajectory generation phase begins for the chunk.

Trajectory Generator

The first step of the trajectory generation is to determine the achievable entry and exit speeds to each corner under the constraints imposed by neighboring corners. For example, if a very obtuse and a very acute corner were placed directly next to each other, it could be impossible to accelerate between the locally achievable cornering speeds for each corner in the distance available under the given kinematic constraints. Sencer et. al. use an iterative process in which block feed rates are raised iteratively until raising a given block's feed rate would result in a violation of the kinematic constraints, at which point the block's feed rate is frozen [34]. However, the velocity incrementing method used by their algorithm can result in substantial undershoots of the truly achievable cornering speeds, especially

since it is applied equally to all blocks. In an improvement to this method, the velocity increment is instead calculated as follows:

$$dV = 0.1 * V + 1\,{}^{mm}/_{S}$$

Thereby limiting the error in the maximum achievable cornering speed to at most 1 mm/s plus 10% of the true maximum while still allowing quick convergence. Additionally, this equation is much more intuitive to modify to achieve a different balance between convergence rate and cornering speed optimality. Rather than operating on a parameter which ties into the overall block's trajectory planner as in Sencer et. al., this iterative process is instead applied to neighboring corners' entry and exit speeds (assumed to be equal to simplify this process), such that these maximum entry and exit speeds are stored along with the corners' entry and exit positions. If a non-zero instantaneous speed change is permissible, it is added to the found entry and exit speeds at this stage.

Then, the time law for the corner (or the assignment of time values to the corner's interpolated path) is computed. This is done using the time-optimal, jerk-limited trajectory planner proposed by Soon et. al., which allows for prescribed entry and exit speeds in its boundary conditions in addition to speed, acceleration, and jerk kinematic constraints [47]. While more complex algorithms exist to compute more general trajectories that allow for prescribed acceleration values at the boundaries such as the numerical methods proposed by Besset et. al. and Biagiotti et. al., they entail substantially more computation and the overall completion speed gained by allowing non-zero acceleration boundary conditions is low, with the benefits of these algorithms being their flexibility for use in higher-order motion such as snap-limited trajectories [48, 51]. A half trajectory is computed using half the total path distance traveled in the corner, the iteratively found corner entry speed as the entry speed,

and the maximum cornering speed, calculated before based on the corner's maximum curvature, as the exit speed. If this exit speed is not reachable under the kinematic constraints, then it is iteratively lowered until it can be reached. To reduce the required computation, symmetry is used to complete the second half of the corner trajectory, such that the corner's exit speed is equal to its entry speed, significantly simplifying the overall trajectory generation algorithm. The time vector for this trajectory is then fitted to the corner's path based on a cumulative path distanced traveled at each point to complete the trajectory of the corner.

After the corner trajectory is computed, straight-line trajectories are computed between the neighboring corners' entry and exit positions and speeds to stitch together an overall trajectory for the chunk. This process is then repeated for all chunks in the print job, completing the trajectory generation for the print job. This results in an overall path and trajectory planning algorithm which is capable of computing near-time-optimal trajectories with a full five controllable input parameters – 4 kinematic constraints (maximum speed, acceleration, jerk, and instantaneous speed change) and one path constraint (cornering tolerance). An example of a completed trajectory is shown plotted below in Figure 4 for the popular Benchy 3D model [55]. Extrusion moves are shown in red, while travel moves are shown as dashed black lines.

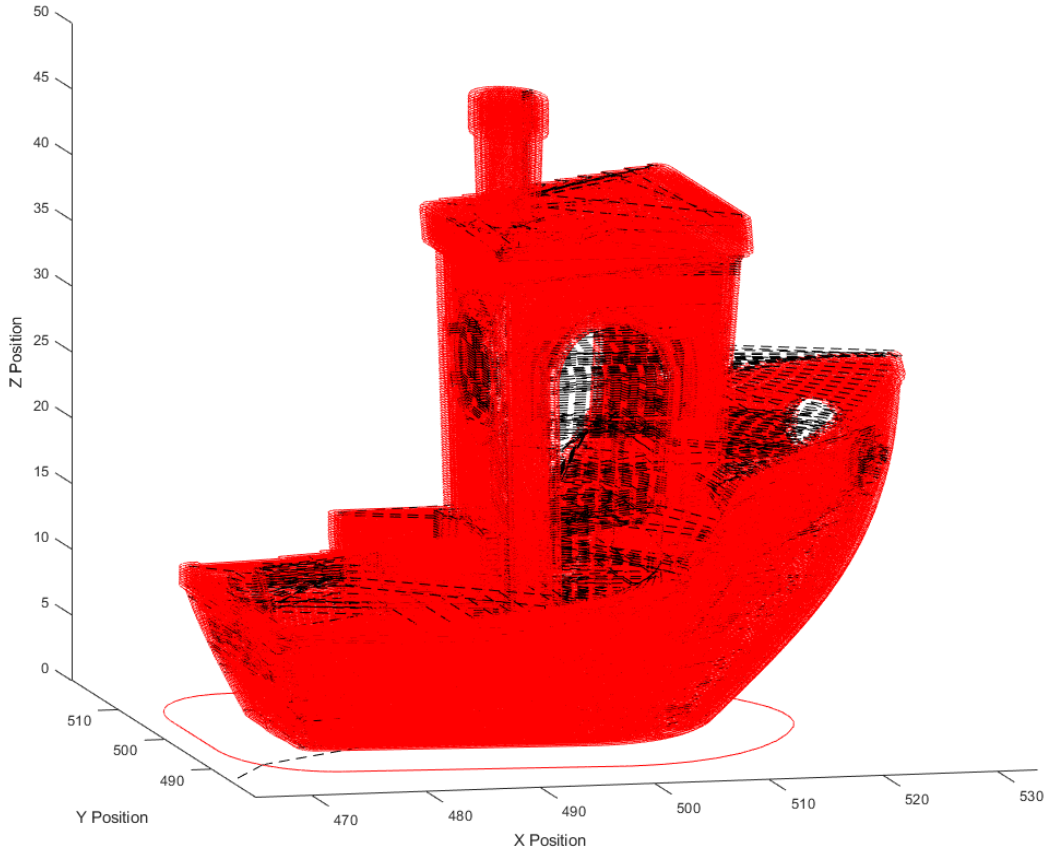

Figure 4: A plot of the trajectory generated for the popular Benchy 3D Model

Hardware Communications Method

For this trajectory to be executed on motion hardware, it must be communicated to the stepper motor drivers, which accept step and direction signals. While the timings for these steps could be computed by the microcontroller in real-time, such real-time interpolation and pulse timing can be arduous for low-cost control boards to compute. In fact, even when using simpler acceleration-limited trajectories, such low-cost equipment is typically only capable of step rates of at most several hundred kHz [32]. Therefore, lower-

level communications are required. To achieve this, most of the computation will be done on the hosting higher-level computer (for example a Raspberry Pi). This will include the g-code parsing, path planning, and trajectory planning, including the computation of the entire step and direction pulse timings. By vectorizing this code, it can be computed quickly on higher-level processors like the Raspberry Pi, leaving the microcontroller to control only the truly real-time portions of the pulse execution and hardware monitoring control loops such as the temperature and fan control.

The pulse and direction timings are computed by interpolating the trajectory at a frequency computed such that the position will change by a step resolution at most every two cycles, allowing one cycle to set the step pin high and another to set it low again in time for the next pulse. A single byte can contain the step and direction line states for all three cartesian axes plus one extruder axis, with further extruder axes requiring two bytes per cycle. Then, the microcontroller can simply directly transfer the received byte(s) to its pin output registries each cycle. This exchanges the high computation requirements for instead high data throughput requirements, which is available in much greater quantities on microcontrollers with hardware USB controllers. Herein, the teensy 4.1 microcontroller is used for pulse output as it has a hardware USB controller allowing for full USB-speed serial emulation while costing less than 30 US dollars, placing it well within the price range of low-cost 3D printers and the open-source community at large. Stepping rates of up to 8 MHz were verified with this method. While even higher rates may be possible, they were not evaluated as 8 MHz is already much faster than most current stepper motor drivers can accept. This is over an order of magnitude improvement over the stepping rates of the current state of the art in open-source, low-cost 3D printer control [32]. With powerful enough motors, this can

allow for much faster and/or much higher resolution motion than was previously possible at this cost level of hardware.

Using code vectorization and bit masking, this interpolation is done highly efficiently, creating a raw binary file containing the pulse train. This file is sent to the microcontroller, which then directly executes these pulses at the fixed frequency computed earlier.

Future Work

While the current work is sufficient for computing near-time-optimal trajectories with five controllable parameters, there are some inefficiencies that could be improved to optimize the balance between optimization of completion time and optimization of computation time. Specifically, the velocity increment equation should be optimized to determine the best trade-off to maximize the computed cornering speeds while maintaining an acceptable computation time. Additionally, the corner time law can be further optimized by allowing for different entry and exit velocities at the corners. Although this would complicate the algorithm significantly, it could yield substantial and worthwhile benefits to motion completion times. To optimize motion completion times even further, other time law approaches should be studied for this application, such as the fully generalized time-optimal approach proposed by Zhang et. al. [56]. Although such an algorithm may prove too computationally expensive to implement on the low-cost hardware discussed here, if it can be made to run quickly enough then it could yield improved motion completion times over the trajectory planner discussed here.

Finally, a significant barrier between the research in the current work and application by the open-source community is the programming language used. Matlab, along with a single python function, was chosen as the programming language to speed up the

development of the algorithm. However, Matlab is inherently a closed-source language. While the open-source language Octave exists which can execute much Matlab code as-is with only minor modification, the algorithm presented here should be re-written in a fully open language. For this, I respectfully suggest Julia as a candidate language for this, due to its combination of ease of programming with exceptional performance.

Conclusion

In conclusion, I have presented a fully integrated path and trajectory planning approach combining and improving upon existing algorithms in the field. My algorithm can accept five input parameters to control the generation of the path and trajectory: the maximum speed, acceleration, jerk, instantaneous speed change, and cornering tolerance. The approach includes a g-code parser and so is directly compatible with the g-code exported by modern slicers and can generate the path and trajectory for extensive g-code files generated for complex objects such as the popular Benchy model. Most importantly, this method is highly computationally efficient and can be implemented on low-cost hardware available to consumer level 3D printers designed by open-source, rep-rap style companies and open-source community designs. Due to its ability to quickly generate trajectories with many controllable parameters, it is also suitable for large data-based studies on motion speed and error, which will be expanded on in the next chapter of this work.

Chapter 2 – Contribution of Path and Trajectory Parameters to Overshoot and Execution Speed in FFF Additive Manufacturing Systems

Introduction

A critical challenge in the additive manufacturing field is the tradeoff between part quality and part production speed. While higher accelerations, motion speeds, jerk values, and more are highly desirable for the reduction of part completion times due to their independence from part resolution, in low-cost systems increasing these parameters can lead to significant vibration being introduced to the motion. This reduces the overall part quality, especially around sharp corners. When tuning these parameters, currently a trial-and-error approach is used. However, this is not a quantitative approach and can be extremely time consuming. Therefore, it would be beneficial to study the effects of various motion parameters on the vibrational error and on part completion times.

Another application that relies on data capturing these relationships is data-informed control, especially machine learning-based techniques. This has recently become an emerging subject in many fields, driven by major advances in the quantity of data that can be accessed and the speed at which it can be processed. Computing advancements have made possible an entirely new aspect of path and trajectory planning in which data is used to either inform or directly assemble the path or trajectory. The methods by which this data can be integrated to path and trajectory planning has become a diverse and growing field that shows great promise in all areas of path planning, from flight control applications to serial robotic manipulators.

Herein, I gather and present a large amount of data capturing the relationship between five motion parameters and the resulting vibrational error and part completion times. The analysis is structured in a way to allow machine learning applications to classify motion types with the goal of determining the optimal motion parameters for the specific type of motion occurring. Importantly, this approach to error and speed optimization is controls-agnostic, allowing for additional filtering or other controls techniques to be serially applied to further suppress vibrations. To allow for others to repeat this approach for other systems, the code used to gather and analyze the data is shared on an open-source basis.

Background and Literature Review

Despite the importance of the quality/speed tradeoff and the recent rise of data-informed control methods, to date no significant databases are available showing the effects of motion parameters such as acceleration and jerk on important results like vibrational error and part completion times. This may be due in part to the difficulty in gathering and analyzing large quantities of data, and the idea that the problem is better solved by simulation than by testing to avoid the time requirement of physical testing. However, simplifying machine models for simulation can overlook aspects affecting the system's performance, especially non-linear effects such as friction, non-newtonian lubrication on motion components, and the non-linear rotational stiffness associated with stepper motors.

Although focused on a real-time controller, Dai et. al. investigates the relationships between trajectory derivatives and the tracking error, and does some physical testing on an air bearing-based motion system to verify a mathematical model of tracking error [57]. For DIW-based processes, Buj-Corral et. al. investigates the effects of printing parameters on the final dimensional error and surface roughness of prints but does not investigate tracking

error or completion times. Buj-Corral et. al. also investigates the effects of various printing parameters on surface roughness and porosity in FFF 3D printing as well as their research on DIW processes [58].

In recent years, machine learning has seen novel use in path and trajectory planning, in which machine learning is used to determine the path and trajectory (sometimes simultaneously) of the machine tool [38-44]. Deep learning and deep reinforcement learning models have dominated this subfield, yielding promising early results. Applications of this method are not limited to machine tools – in fact, machine learning has been applied to trajectory generations even in the field of air traffic control algorithms in Guan et. al. [38]. Even among more classical machine control applications, these machine learning methods approach the problem of path and trajectory generation from a variety of angles and for a variety of systems. Zhou et. al. apply machine learning to train an inverse kinematics model by generating learning-ideal spline-based trajectories to generate the dataset used for the machine learning model in a structured way to out-perform more intuitive trial-and-error approaches to dataset generation used by previous machine learning-based algorithms [41]. Machine learning has also been applied to tracking error prediction by Wu et. al., who use deep learning to predict tracking error in 2-axis machining [43]. Rather than study the impact of motion parameters, they study the impact of the g-code being executed and train their model using randomly generated g-code commands.

To support the application of machine learning in path and trajectory planning algorithms in the field of additive manufacturing, large amounts of data is required relating the vibrational error and part completion time back to the controllable motion parameters. If enough data is gathered for a particular machine, then a relatively simple classification model could be used to determine the type of motion and then simply select the combination

of motion parameters resulting in the most desirable combination of vibrational error and overall motion speed.

Methods

Typically, overshoot error is evaluated by applying a step motion on the axis. However, in this case there is a path planning component that involves a corner smoothing algorithm. Therefore, to get a step motion to measure while still including the effects of the path planner, the motion computed must consist of multiple moves at right angles to each other to form a rectangle in the XY plane. When outputting and measuring only a single axis of this move sequence, it appears as a series of step motions with pauses in between, which is convenient for analysis while still capturing the effect of the corner smoothing path planner.

To characterize the completion rate of a given set of settings requires more complexity than merely considering the completion time of the overshoot error testing. Since it is possible that settings will perform differently for different waypoint topologies, a variety of topologies should be considered. To do this, I propose the categorization of cartesian motion in 3D printing into three categories: corners, characterized by several long motion segments at steep angles to their neighboring segments, curves, characterized by many short motion segments at shallow angles to their neighboring segments, and raster fill, characterized by alternating long and short segments at right angles to each other consistent with the raster fill patterns. To create realistic testing g-codes for these, 3D models were generated with the intent of idealizing the model to produce a toolpath consisting of mostly or only moves of one of these categories. The models as designed are shown below in Figure 5:

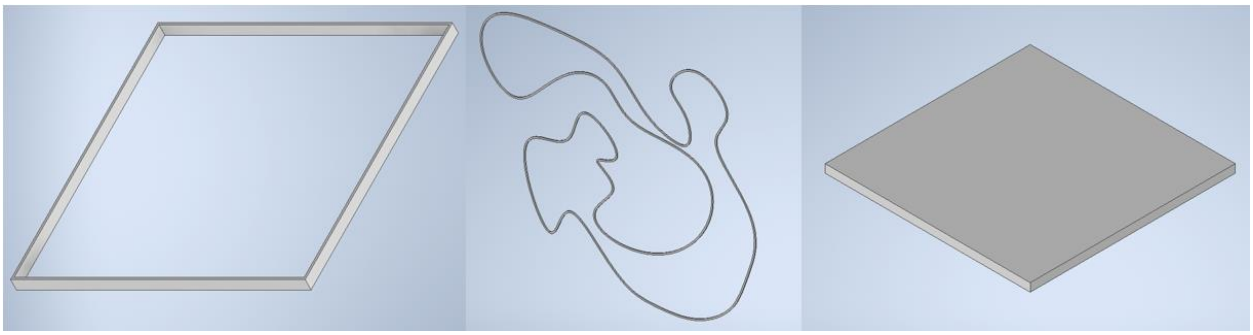

Figure 5: Characteristic objects. From left to right: the corners object, curves object, and raster fill object.

These characteristic objects can then be sliced in any open-source slicer, resulting in g-code for each object. Trajectory generation is then done for these g-codes, and the completion time of the resulting trajectory is taken as the characteristic time for that object for the given set of settings. This is repeated for all sets of settings evaluated, and the times then normalized against the mean completion time for that object to be comparable across different characteristic objects.

Trajectory Generator and Independent Variables Studied

For trajectory generation, the path and trajectory planner described in Chapter 1 of this thesis is used. This allows for the control of five variables: velocity (V), the maximum allowable linear motion speed of the toolhead, acceleration (A), the maximum allowable linear acceleration of the toolhead, jerk (J), the maximum allowable jerk of the toolhead, cornering tolerance (cTol), the maximum allowable deviation of the planned path from the waypoints in the g-code, and instantaneous velocity change (instV), the maximum allowable instant change in the linear motion speed when entering and exiting corners. Therefore, the independent variable space has a dimensionality of 5. Due to this dimensionality, it is difficult to physically test enough settings to properly sample the variable space. To deal with this, the variable space is sampled unevenly, with the V, A, and J being sampled fairly

densely while cTol and instV are sampled very coarsely. Each combination of these sampling points is then tested as a set of settings. The values tested for each individual setting are shown below in Table 2, for a total of 6000 combinations of settings to be tested:

Table 2: Sampling points

V (mm/s)	A (mm/s²)	J (mm/s³)	cTol (mm)	instV (mm/s)
50	1000	100000	0	0
100	2000	250000	0.025	10
150	3500	500000	0.05	20
200	5000	750000	0.10	
250	7500	1500000		
300	10000			
375	13000			
450	16000			
525	20000			
600	25000			

Physical System and Data Collection

To test the motion, a single-axis prototype was constructed with the goal of simulating a typical FFF 3D printer belted motion axis. To eliminate as many external variables as possible, the system was built on a base of 3/4" thick aluminum, Blanchard ground on both sides. A single, oversize linear rail (Misumi) guides the motion, and the belt is located directly above the linear rail bearing, minimizing resultant twisting moments on the carriage. Steel weight blocks were used to simulate the approximate moving mass of a typical low-cost FFF 3D printer toolhead, and an extremely high-power NEMA 24 stepper motor and driver (Oriental Motor) were used to drive the system, allowing for extremely high motion speeds and accelerations. Finally, an incremental optical linear encoder (Renishaw) was used for position capture, with a resolution of 20 nm and sub-micrometer absolute accuracy, guaranteeing highly accurate data collection. The step and direction signals were sent by a

microcontroller (Teensy 4.1) as described in Chapter 1 of this thesis. Images of the assembled motion stage is shown below in Figure 6. This setup is then clamped to a large lab bench to further suppress external vibrations. The Renishaw optical encoder can capture motion at up to 725 mm/s, safely above the highest tested setting for V.

The rectangular trajectory for testing overshoot error is tested 5 times in sequence for each combination of settings, recording the resulting motion trajectory. Reading of the optical encoder is accomplished via a high-speed USB Data Acquisition system (DAQ) (National Instruments), and the resulting counter data converted into position data. Since this amount of data would be intractable to collect by hand, this process was automated in 10-11 4-hour intervals, allowing the large number of data points to be feasibly collected.

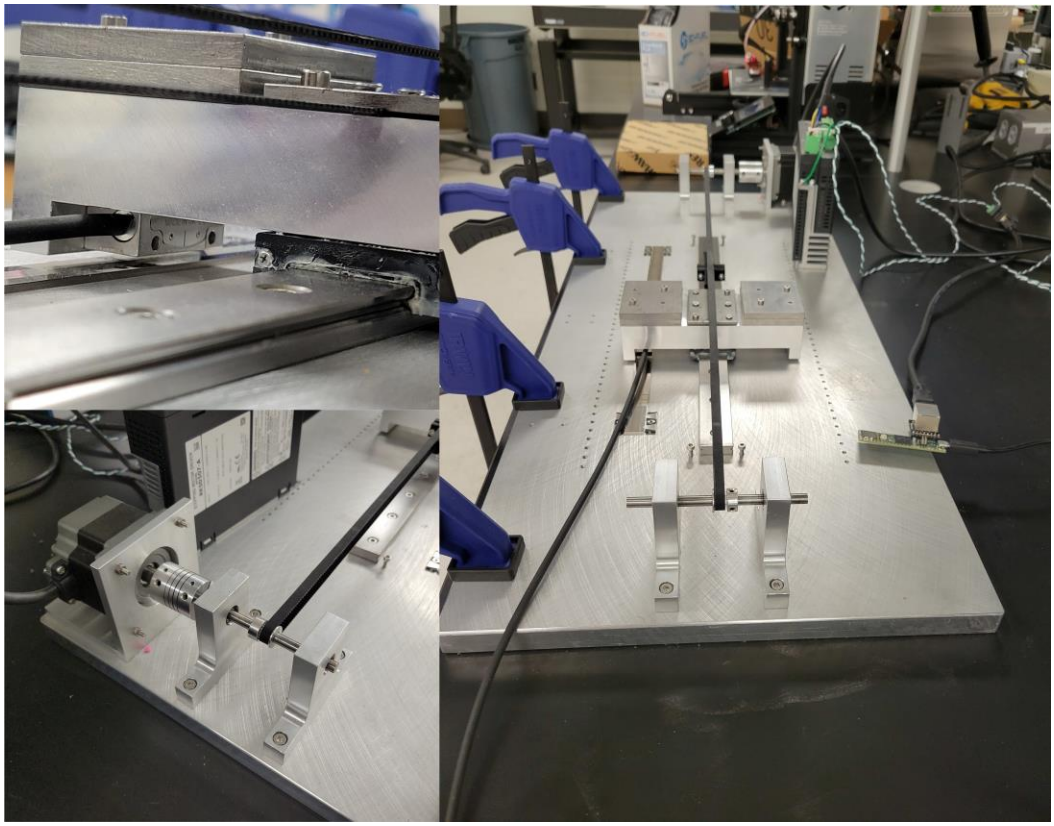

Figure 6: Physical system setup.

Analysis Methodology

First, the overshoot error is analyzed in the collected data for the overshoot error testing. The analysis code automatically detects each of the 5 repeated tests within the trajectory and begins by breaking the trajectory up into these separate tests. Then, the overshoot error was calculated using the maximum and settled positions for each of the 5 tests, and the results averaged to determine the final overshoot error. In cases of an error in the data, e.g., due to USB disconnects or other operating system processes interfering with the data transfer, the code is equipped to automatically flag suspicious test runs to be retaken.

After all such test runs are fixed, the analysis then proceeds to compute the trajectories for the characteristic objects for the test runs' settings, storing the resulting completion times alongside the overshoot error for the test run within the motionTestRun() class described in Chapter 1 of this thesis. After this process is completed for every test run, for each characteristic object and resulting trajectory these completion times are normalized against the average of all of the completion times so that they can be compared across different characteristic objects. From there, any desired Figure of Merit (FOM) can be easily calculated based on any/all of the available parameters of the test run. For example, one such FOM would be a normalized motion speed FOM calculated as follows:

$$FOM = \sqrt{\left(\frac{1}{NS_{CN}}\right)^2 + \left(\frac{1}{NS_{CR}}\right)^2 + \left(\frac{1}{NS_{RS}}\right)^2}$$

Where NS_{CN}, NS_{CR}, and NS_{RS} are the normalized completion times of the corners, curves, and raster fill characteristic objects, respectively, and a higher FOM corresponds to faster motion. This is a powerful tool, as a tailored FOM could potentially weight the characteristic object

completion times in future works based on the geometric topology of the current motion chunk.

Results

Effect of Velocity on Accel/Decel Phasing

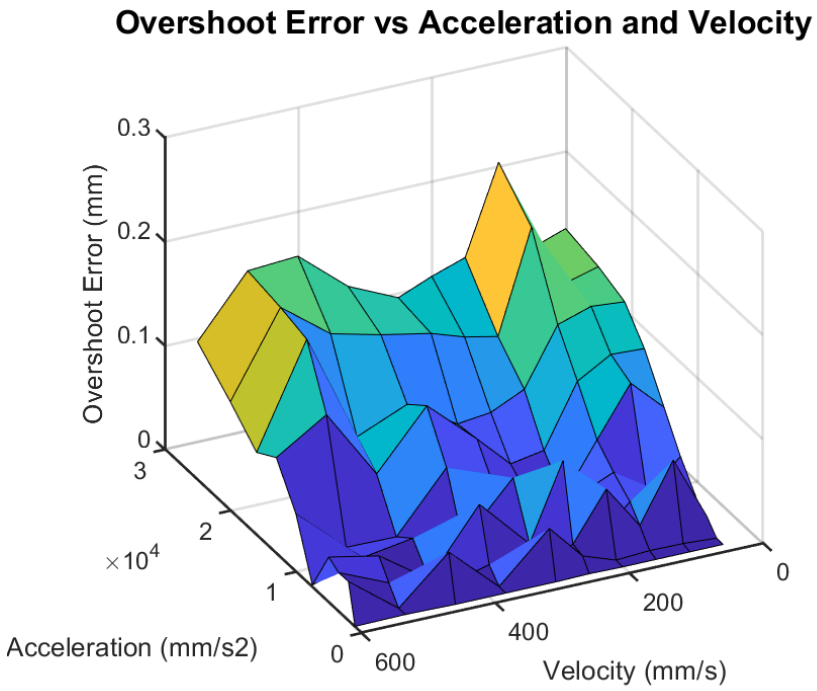

Figure 7: Overshoot error plotted as a function of velocity and acceleration. Notice that while there is a clear trend along the acceleration axis, the velocity axis on the other hand appears to be only noise. This is an indication that the maximum speed actually has minimal effect on the vibratory error.

Importantly, this plot shows that there is no significant trend along the velocity axis – it simply appears to be adding noise to the results. I theorize that this noise is due to variable phasing between the acceleration and deceleration segments. When these phases are timed exactly right relative to one another, the vibratory error can mostly cancel out, resulting in a form of natural input shaping that occurs and reduces the error for that set of settings. Changing the velocity setting modifies the separation time between the acceleration

and deceleration segments, thereby affecting the resulting overshoot error. It may be possible to use this mechanism to reduce motion error in a more computationally efficient way than the post-processing required by input shaping. Unfortunately, this effect could be exceedingly difficult to reliably exploit, as it is by nature geometry dependent due to variable move length. In fact, for short moves, the maximum speed may have to be set extremely low to exploit the effect at all, resulting in a net negative to the overall motion speed. To test this phasing theory, I changed the plotting code to consider the velocity case with the highest overshoot error for each combination of the other settings of interest. Then, instead of plotting it against velocity and acceleration I chose to plot against jerk and acceleration. Plots of overshoot error are shown below for a cornering tolerance of 0.05 mm and an instant velocity change of 10 mm/s to evaluate the theory, although these choices of settings are not important to the result and were simply chosen as the midpoints of values tested for the respective settings. If the phase angle of vibrations induced by the acceleration and deceleration segments is responsible for the noise seen in Figure 7, then such noise should not be present when considering the worst-case velocity setting for each other combination of settings. The error surface is shown, first for V = 150 then for the worst-case velocity for each data point plotted, in Figure 8 and Figure 9 below. The noise is almost entirely eliminated, and much clearer trends emerge in the error surface. Therefore, this analysis approach is adopted for the remainder of this chapter – at any point overshoot error is used, it is taken to be the worst overshoot error occurring at any velocity for the remaining combination of settings. In this way, the dimensionality of the analyzable data set is effectively reduced by one to account for this phasing effect between the acceleration and deceleration segments.

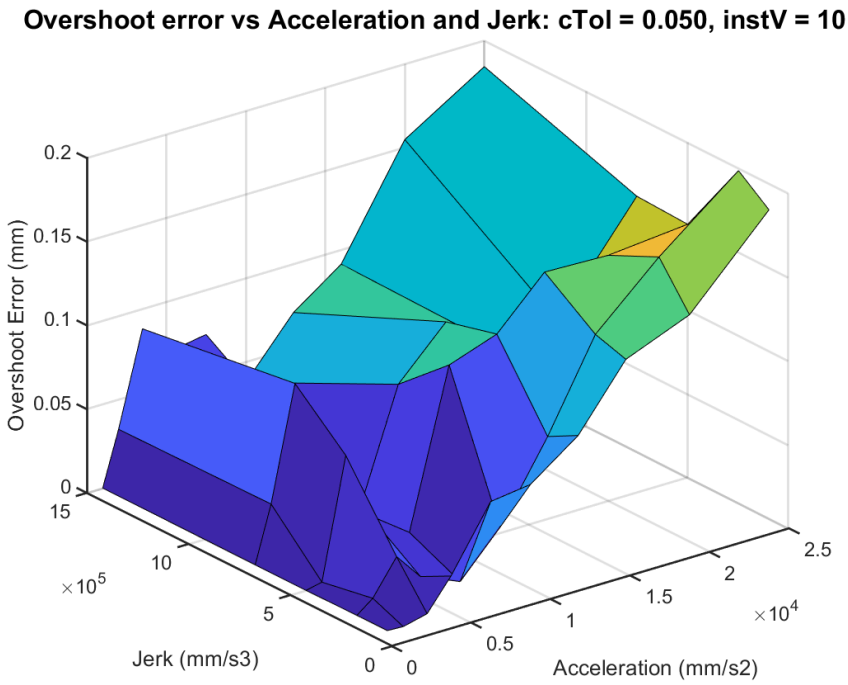

Figure 8: A plot of overshoot error, shown for $V = 100\ mm/s$

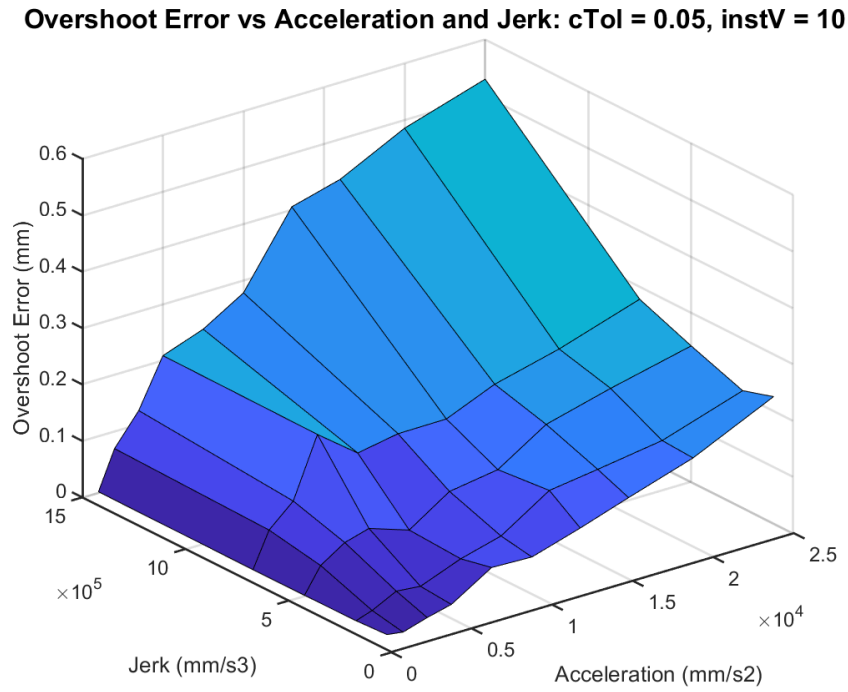

Figure 9: A plot of overshoot error, with each data point determined using the worst-case setting for V.

Overshoot Error

With the noise resolved, the data can now be properly examined for trends and evaluated as a tool for setting optimization. To begin with, to view all the raw overshoot error data, all the overshoot error plots against A and J are shown below. They are organized into 3 sets of 4 plots, with each set of 4 plots corresponding to a specific instV setting, and each of the 4 plots with the group corresponding to each of the 4 cTol settings. In this way, one can easily visualize how the cTol and instV parameters affect the shape of the error surface in the A-J variable space:

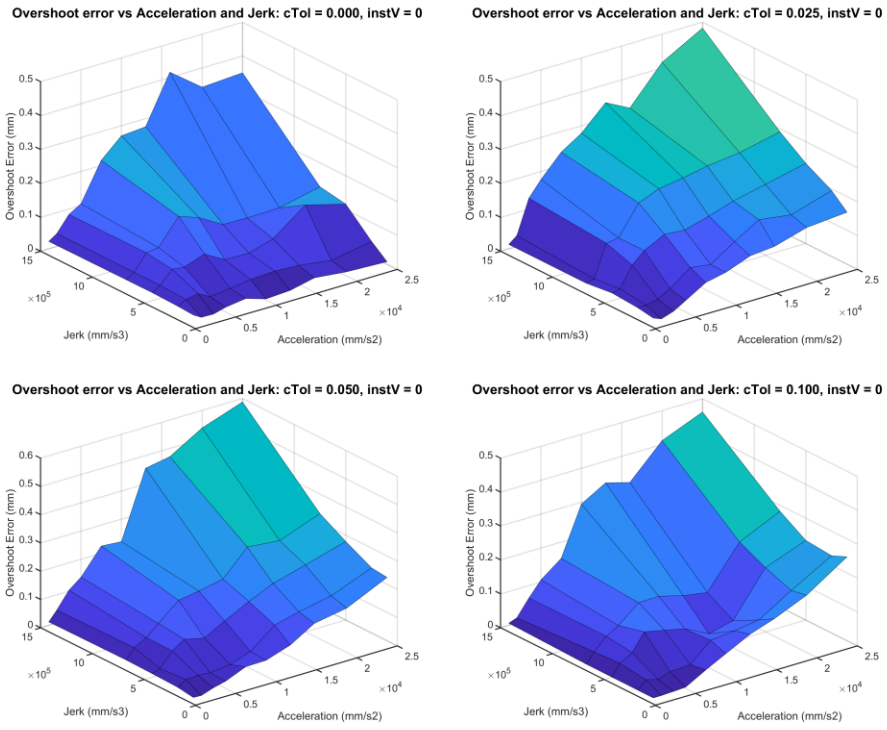

Figure 10: Overshoot error plotted for instV = 0 and various settings of cTol

An interesting feature of the error surfaces is that the use of corner smoothing actually increases the effect of acceleration on the overshoot error. It is not clear why this is the case – possibly, the frequency spectra of the corner trajectories fall nearby the resonant frequency

of the system, which is approximately 44 Hz, in a way which does not occur during the linear moves. More study should be done on this phenomenon to determine its exact cause.

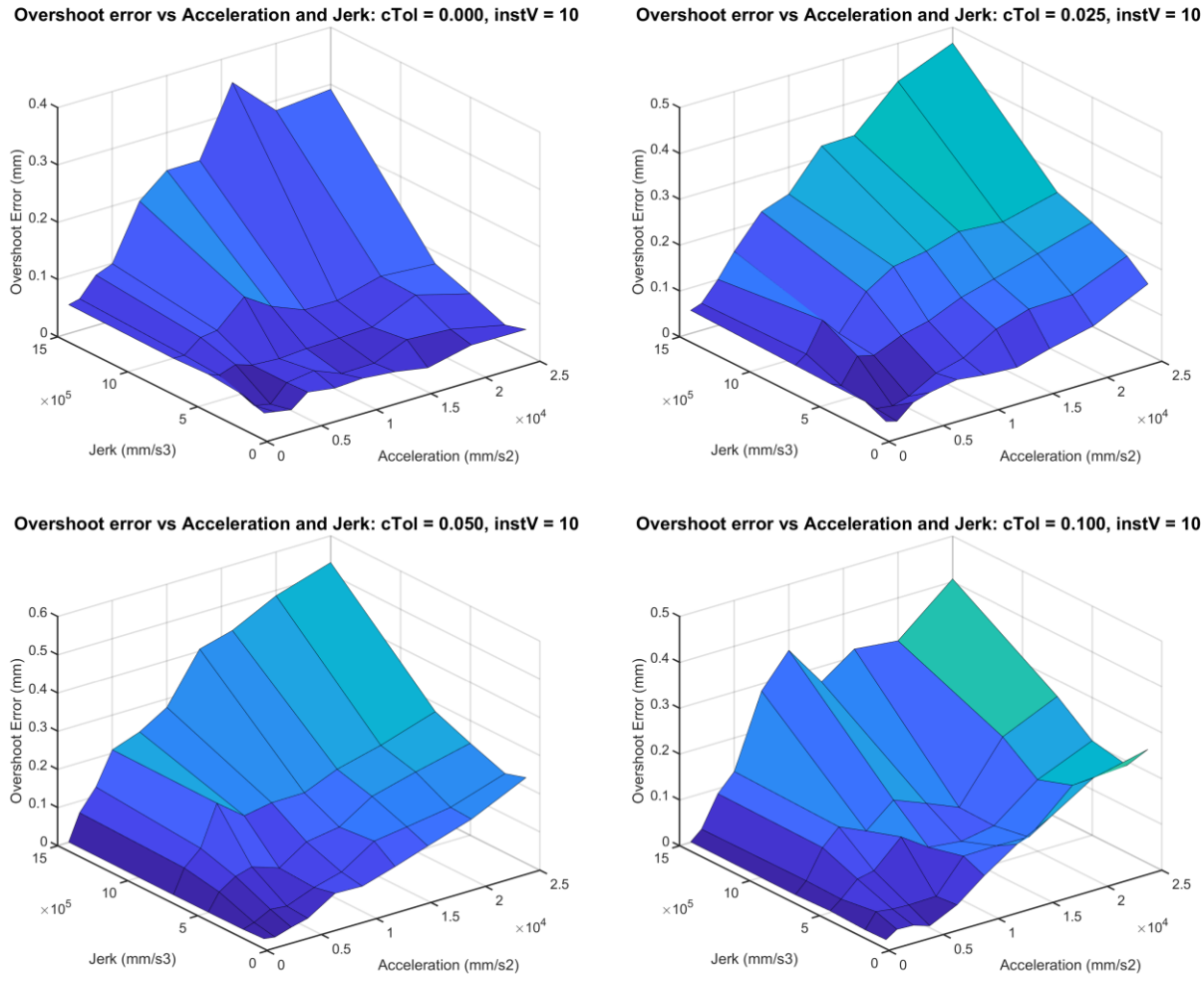

Figure 11: Overshoot error plotted for instV = 10 and various settings of cTol

Interestingly, when adding in an allowable instantaneous velocity change, the overshoot error actually decreases nearly across the board. This runs counter to the expected result, as it would be expected that these velocity discontinuities would increase vibrational overshoot in the response, not decrease it.

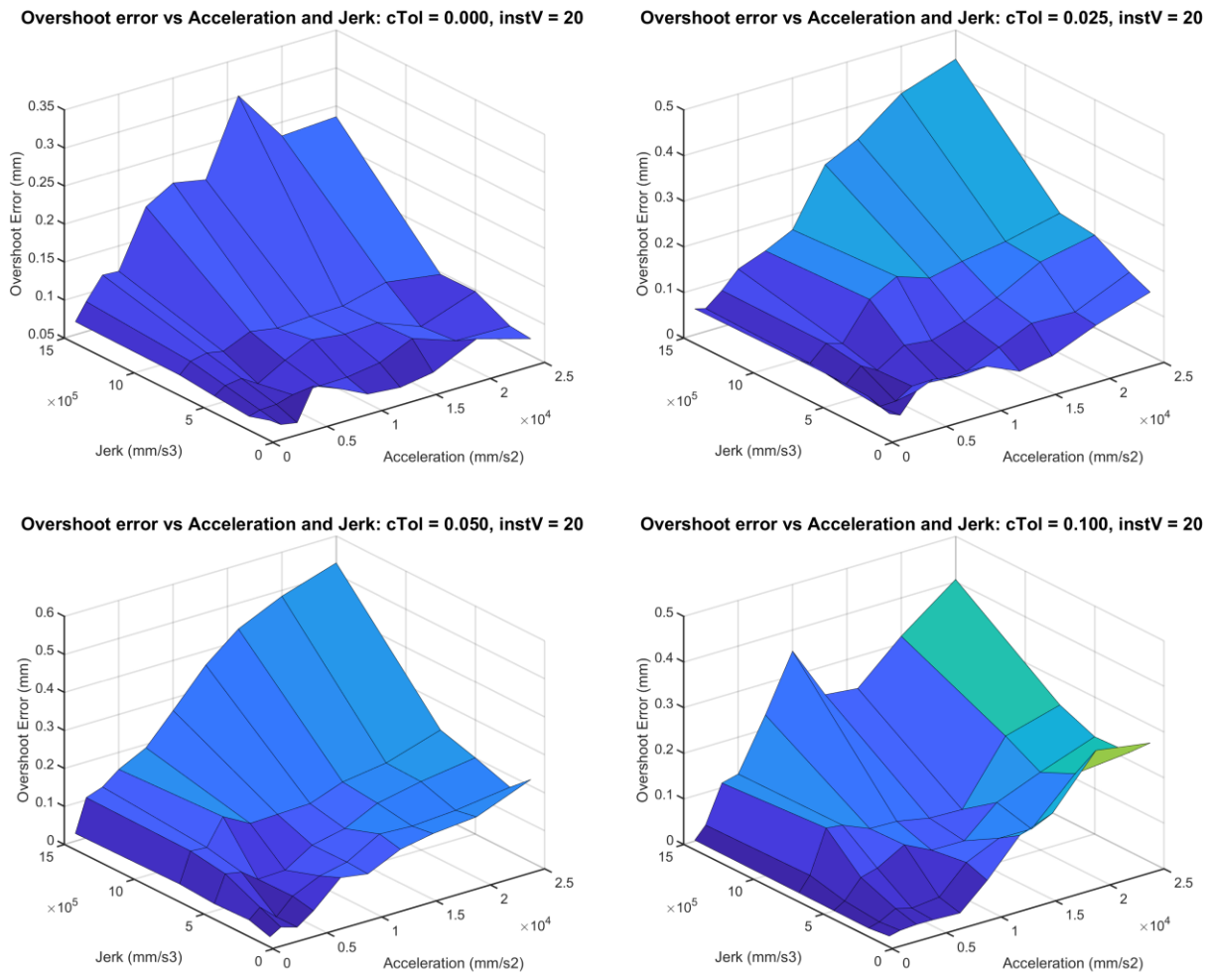

Figure 12: Overshoot error plotted for instV = 20 and various settings of cTol

When increasing the allowable instantaneous velocity change to 20 mm/s, the overshoot error continues to decrease across the board, indicating a trend rather than a fluke. This phenomenon could prove to be an important one if it can be consistently exploited by investigating the cause.

Example FOM Plots

In addition to the error surfaces, it is of course possible to compute and plot any figure of merit (FOM) based on the available data described in the methods section. One example of a FOM is one which is zero when the overshoot error is above a tolerance value and equal

to the normalized movement speed when it is below a certain value, with the normalized movement speed computed as shown in the methods section. Plots of such a figure of merit for a tolerance value of 0.2 mm are shown for instV = 0 mm/s, cTol = 0 mm and 0.025 mm below in Figure 13 and Figure 14, respectively:

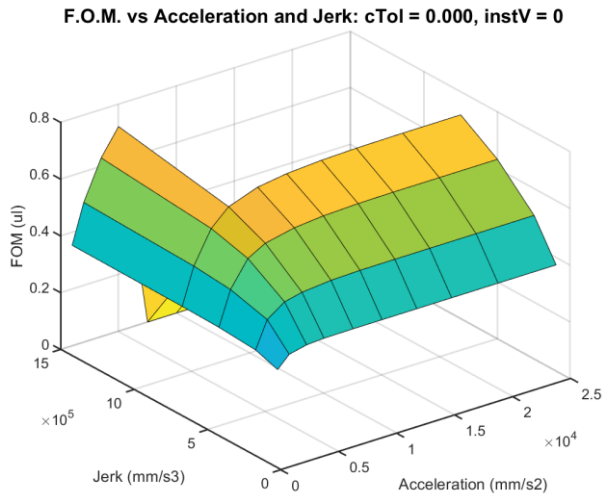

Figure 13: Normalized motion speed FOM plotted against Jerk and Acceleration for cTol = 0 and instV = 0.

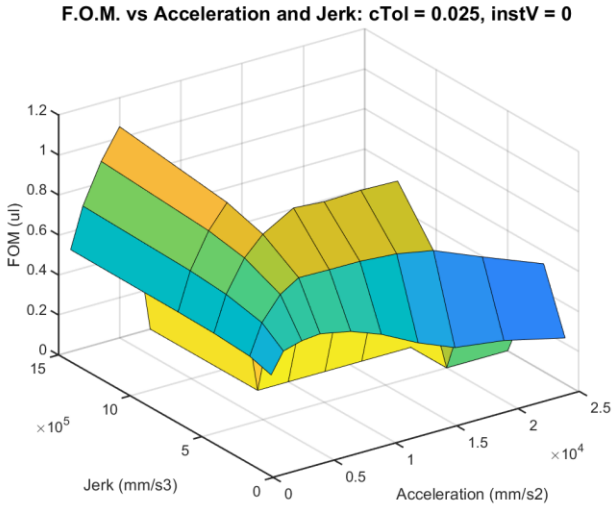

Figure 14: Normalized motion speed FOM plotted against Jerk and Acceleration for cTol = 0 and instV = 0.

These figures show that adjustments to ancillary settings like the cornering tolerance and instant velocity change can change the normalized motion speed by a large amount, with cTol = 0.025 mm resulting in approximately 40% faster overall motion speeds compared to cTol = 0. These types of effects can make it extremely difficult to adjust motion settings by trial and error, as they represent complicated inter-variable relationships that cause constant re-adjustment of prior settings.

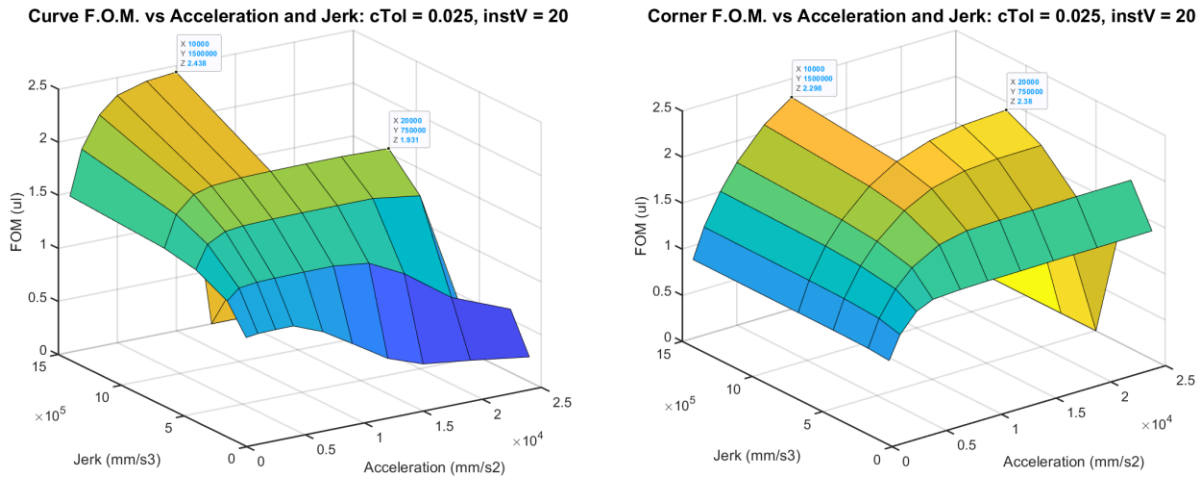

Figure 15: Left - FOM for curve speed, right - FOM for corner speed.

Whether an object comprises curves or corners has significant implications to the ideal settings. For curves, it is better to use high jerk but more moderate acceleration, while for corners it is better to use higher acceleration but lower jerk. This indicates that there is a strong reason to believe that this data-based analysis approach can be extended with machine learning in the future to achieve optimized settings for every section of an object via machine learning classification of motion types and assembling an appropriate FOM formulation to find the optimal settings given the classification weight results.

Future Work

In future works, it is recommended to investigate more thoroughly several of the phenomena seen in the data collected. More work should be done to analyze the variable phasing effect that appears to be controlling the noise injected by the velocity setting, as if a better balance between noise reduction and the necessary number of velocity points sampled can be achieved, this could allow for higher sampling resolution in other settings. Another important phenomenon seen in the results is the higher overshoot error exhibited by higher settings for the cornering tolerance, and if it is dependent on the angle of the g-code turn or if it is the result of some other effect. In addition to the unexplained phenomena, work should be done on classification of g-code, whether by machine learning or by a different approach. This could allow the use of datasets such as the one discussed here to be applied toward the optimization of motion settings at every portion the part to fully optimize the part completion time and vibrational error.

Research should be done toward how optimal parameters should be constructed in a machine with significantly anisotropic characteristics, such as machines with large inter-axis differences in error surfaces. More open-source data should be captured and published, especially for common open-source 3D printers, for which a single dataset could allow for easier tuning of settings by all operators of that 3D printer. Finally, more work should be done to determine if the trajectory generator's corner time law can be further optimized without costing too many computational resources to be run in real time on a single board computer.

Conclusion

I have presented a first of its kind dataset relating several motion parameters to their effects on vibrational overshoot error and shown that significant opportunity exists for optimizing the processing speed of parts manufactured by Fused Filament Fabrication using g-code classification with either machine learning or even using a formulaic approach. Code was written and implemented in Matlab to automate the bulk of the data collection and processing, allowing for enough data points to be gathered and analyzed to sample a five-dimensional motion parameter space well enough for visualization of the effects of each variable on overshoot error surfaces. Several phenomena are observed in the error surfaces, including an apparent relative phasing effect between the acceleration and deceleration motion segments causing vibrational error to interfere constructively and destructively. Several avenues for future research exist, especially in g-code classification to provide chunk-level settings optimization to reduce part production times while balancing the vibrational overshoot. Other future research areas include investigation of the phenomena discussed and attempts to further optimize the balance between the optimality of the cornering time law and its computational cost.

Chapter 3 - 3D Printing Autoclavable PPE on Low-Cost Consumer 3D Printers

Introduction

The onset of a global pandemic has revealed a need for widely distributed manufacturing of medical Personal Protective Equipment (PPE). With demand for PPE spiking and traditional manufacturing unable to scale production quickly and flexibly enough to match it, medical facilities around the world began using 3D printed PPE sourced from their own 3D printers, local makerspaces and universities, or even individuals who own one or more 3D printers [59]. Previously, work to bring a distributed supply chain to autoclavable PPE has focused on creating new 3D printer designs capable of printing high-temperature thermoplastics, such as the open-source Cerberus 3D printer from Michigan Tech [60]. There has also been some analysis done on design efficacy of individual PPE designs printed on commercially available printers, but little to no research has taken place on expanding the printing materials used by these commercial printers, which presently is typically Polylactic Acid (PLA), Polyethylene Terephthalate Glycol (PETG), Acrylonitrile Butadiene Styrene (ABS), Acrylonitrile Styrene Acrylate (ASA), or nylon [61, 62].

While easy to print, PLA has a Vicat softening temperature of around 62 °C [63]. This means it cannot be sterilized by steam autoclave, which takes place at 121 °C, and must be sterilized with other methods such as surface disinfectants to avoid severe warping and drooping of the part. While larger facilities such as hospitals are likely to have other compatible automated sterilization methods such as low-temperature hydrogen peroxide gas plasma [64], smaller or older facilities without such modern equipment available currently

need to manually disinfect 3D printed PPE, which can take medical personnel several minutes per item [65]. Steam autoclave is a disinfecting process which uses pressurized steam at 121 °C to sterilize objects by way of heat [66, 67]. It is an effective and automated sterilization process and allows for objects to be sterilized in large quantities. 3D printing PPE in an autoclavable material would save valuable time for medical professionals on the front lines of the pandemic, as well as building confidence that the parts are fully disinfected, since sterilizing 3D printed parts with surface disinfectants can be inconsistent due to the large number of small gaps and crevices which result from the 3D printing process. To allow steam autoclave to be used as a disinfecting method for 3D printed PPE, it is necessary to expand the materials used to print them to include those which can survive autoclave temperatures without major deformation. Since consumer 3D printers are not capable of high extrusion temperatures and do not have high temperature heated chambers [5, 6, 68, 69], they are incapable of producing parts made from high-temperature engineering thermoplastics that extrude at well over 300 °C, such as Polyether Ether Ketone (PEEK) or Polyetherimide (PEI), which would typically be chosen when autoclavability is required. To save the time of medical professionals, it is therefore desirable to demonstrate the use of an autoclavable thermoplastic with widely available consumer 3D printers with as little modification as possible to provide widespread, autoclavable PPE manufacturing capabilities.

In this paper, we demonstrate the 3D printing of a nylon copolymer (CoPA) manufactured by Polymaker and evaluate its resistance to autoclave conditions by printing and autoclaving test components which consist of large bridges and overhangs to test for unacceptable deformations [70]. Tensile testing specimens with dimensions according to ASTM D638 Type V are also 3D printed and tested on a tensile testing machine (Instron) to

check if and how mechanical properties of the material are degraded by autoclave. The print quality of the CoPA material is raised to acceptable levels by designing a very simple heated chamber, capable of heating to up to 50 °C without causing damage to the printer, consisting of a cardboard and clear plastic sheet enclosure built around the printer with a commonly available 500-watt personal space heater and temperature controller available from Amazon.

Theory and Hypothesis

Common engineering thermoplastics with high temperature resistance also have a high glass transition temperature, with PEEK and PEI having glass transition temperatures of 143 °C and 217 °C respectively [71]. As a thermoplastic cools, once it reaches its glass transition temperature it can no longer release internal stresses, which in 3D printing causes warping of the printed part due to thermal contraction of the polymer [72]. Large differences between a 3D printing polymer's glass transition temperature and the build chamber temperature can therefore cause a large amount of warping, and thermoplastics with a high glass transition temperature, including PEEK and PEI, typically require a high temperature heated build chamber to print successfully. Other factors in part warping include a polymer's thermal expansion coefficient and crystalline behavior, although these behaviors can be more complicated and are frequently not reported by 3D printing filament manufacturers. Some thermoplastics – including PEEK, but not PEI, maintain physical strength at temperatures above their glass transition temperature [71]. One measure of what temperature a polymer softens at is its Vicat softening temperature, making the Vicat softening temperature a simple method of predicting whether a polymer will survive autoclave cycles without deforming [71]. Therefore, to find a 3D printing thermoplastic which will survive autoclave but be printable on a Prusa MK3S or similar machine, the authors hypothesize that a polymer should have a Vicat softening temperature well above the 121 °C of steam autoclave,

a low glass transition temperature, and a recommended printing temperature safely under the 300 °C limit of the Prusa MK3S hotend. Semi-crystalline polymers should be of particular interest due to their different modulus characteristics in the temperature region near their glass transition temperature compared to amorphous plastics [73]. A plot of Vicat softening temperature (VST) vs. glass transition temperature (Tg) of many 3D printing materials is shown below in Figure 16 [71, 74-78].

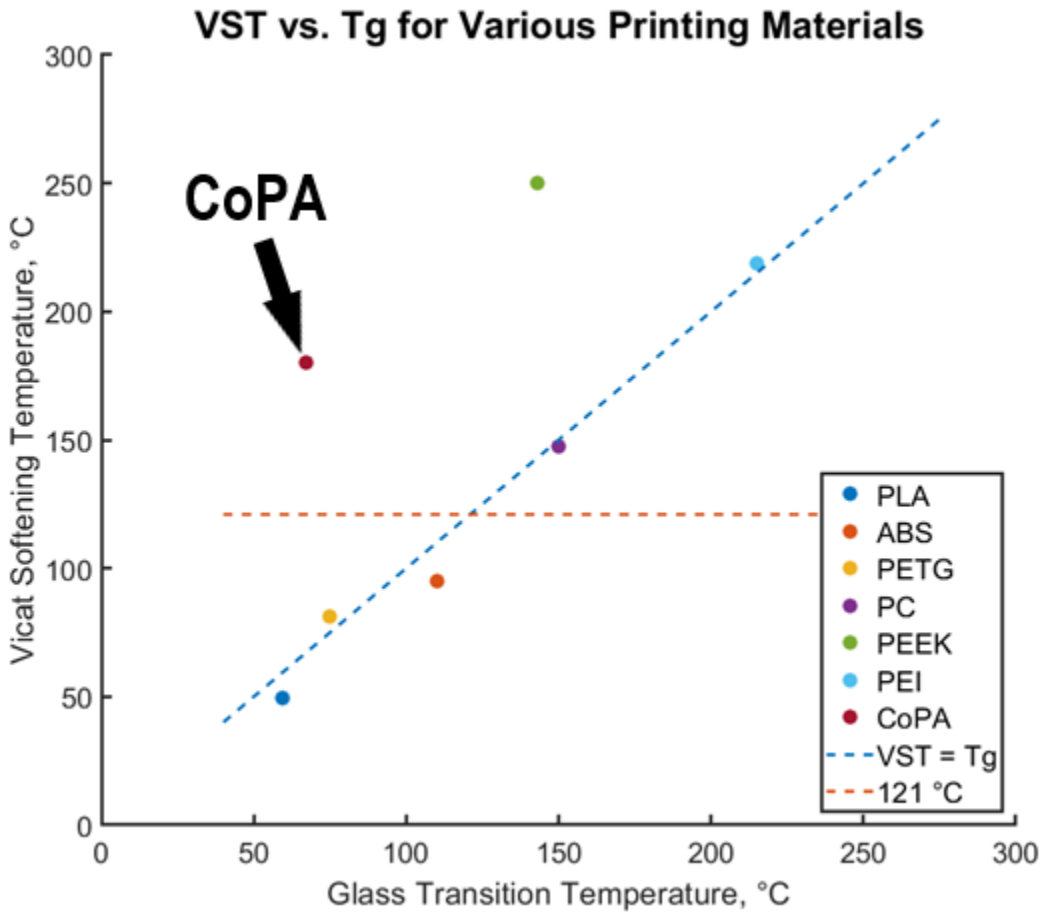

*Figure 16: Vicat softening vs. glass transition temperatures of many 3D printing filament types and blends. Notice that most polymers follow the line VST = Tg, but that notable outliers include CoPA and PEEK. *For PEEK, the continuous service temperature is shown*

CoPA, a nylon-based copolymer made by Polymaker, has a Vicat softening temperature of 180 °C, a glass transition temperature of 67 °C, and a recommended printing

temperature of 250-270 °C, making it a good candidate for testing [79]. The authors hypothesize that it should be possible to print CoPA on a Prusa MK3S or any similar printer, since the manufacturer's recommended build environment temperature is only 40-60 °C [79], and a chamber temperature of up to 45-50 °C should be tolerable for the motors, electronics, and 3D printed components of most consumer machines.

Materials and Methods

Consumer 3D Printer selection

The Prusa MK3S 3D printer was selected to represent a commonly available consumer 3D printer. Available at $800 USD and with a production volume of over 60,000 printers in 2019 [80], it is popular in makerspaces and universities as well as with individual owners, making it widely available for distributed manufacturing of PPE. It has also already been used in other studies on 3D printing PPE [61]. With a maximum extrusion temperature of 300 °C, it cannot extrude high temperature engineering thermoplastics such as PEEK and PEI, but is capable of extruding CoPA which prints at 250 - 270 °C.

Heated enclosure setup

The heated enclosure was constructed from cardboard, tape, a space heater and temperature controller, and clear sheet plastic to provide a viewing window, along with two twist ties to help secure the enclosure door. A 5-sided oversized box was constructed around the printer, with the fifth (front) side of the box hollowed out to an approximately 3-inch frame and copied a second time. The center of the second copy was filled with the clear plastic sheeting to create a see-through door, which was hinged to the front top edge of the enclosure. Finally, a hole was pierced in the top for the temperature probe and a cutout was made in

the back of the right panel to install the space heater. The completed enclosure with all components installed is shown in Figure 17. More detailed assembly instructions are in Appendix 3A.

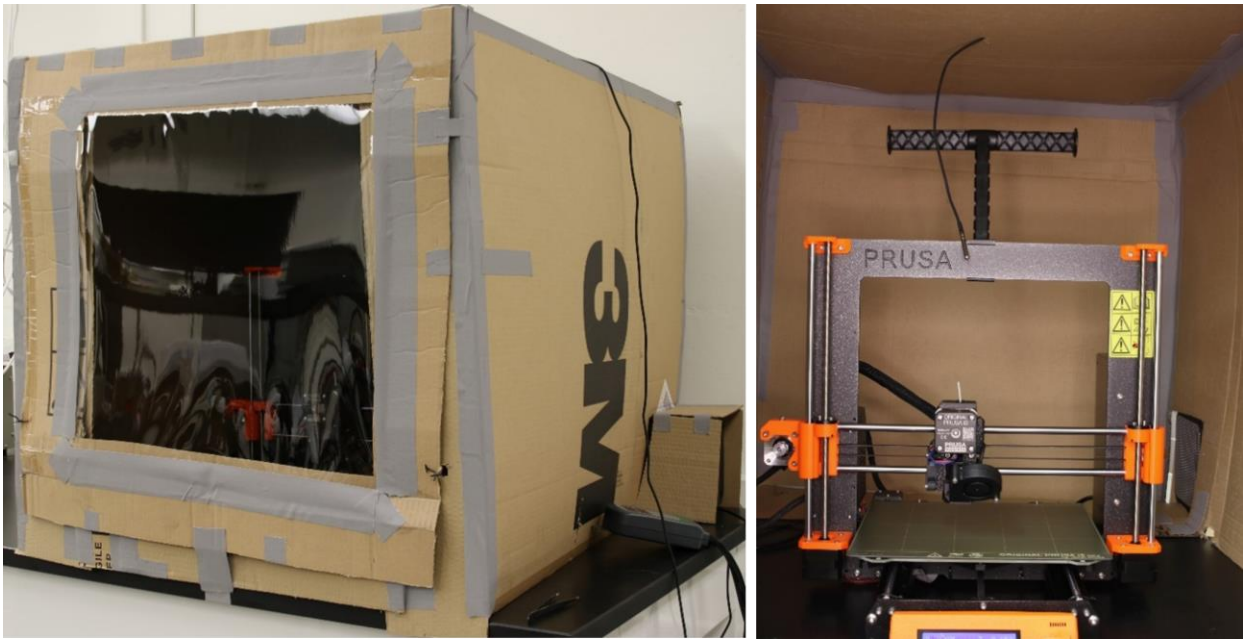

Figure 17: Cardboard heated build chamber. (a): Closed chamber. Note the use of twist ties to secure the chamber door and clear plastic sheet used to provide visualization of the print. (b): Inside of the chamber. Note the position of the space heater (AmazonBasics) and temperature prove of the off-the-shelf temperature controller (Inkbird ITC-308).

While a much higher quality enclosure could be constructed using standard aluminum extrusions and plastic panels for example, the authors' intent is to demonstrate that such engineering time and materials are unnecessary to create a satisfactory heated chamber which functions well enough to raise the print quality of the CoPA material to an acceptable level. The described enclosure can be constructed in only an hour or two with minimal tools, and only requires materials and tools which are commonly available and are likely already owned by most individuals and makerspaces.

3D Printing of CoPA

To initially test for drooping and structural integrity in autoclaved components, test pieces including a tower with 4 increasingly large overhangs and a bridge were printed and subjected to a 121 °C steam autoclave sterilization cycle. Test pieces were then inspected for any substantial deformation and compared to test pieces printed in PLA and Acrylonitrile Styrene Acrylate (ASA). To check for unacceptable degradation in mechanical properties, 20 ASTM D638 type V tensile testing specimens, with a cross-section of 3.2x3.2mm, were manufactured using perimeter-only 100% infill, and half were autoclaved. In each group, 5 specimens were dried prior to testing while 5 were tested while acclimated to atmospheric moisture content. Prior to printing, all filament of each material was dried thoroughly to avoid quality issues caused by atmospheric moisture absorption. During testing, it was noticed that prolonged heat soaks at drying temperatures (several days or more over many overnight drying cycles) were causing the filament to gradually degrade and become brittle and fragile. For this paper, authors used vacuum sealing filament containers with desiccant packs to store and dry the filament to avoid heating it repeatedly but recommend that others make an airtight filament dry-box with Bowden fittings to allow for printing from the dry-box directly. The manufacturer states that CoPA can be abrasive to brass nozzles when used extensively [70], although no nozzle abrasion was noted by the authors during the printing of approximately 1.5 kg of CoPA through an uncoated brass nozzle.

Results and Discussion

Part fidelity and deformation tests

In initial testing, no heated chamber was used, and the print quality with CoPA was consistently poor. Overhangs showed a tendency to curl upward when cooling to room

temperature and the print quality also degraded as the filament absorbed atmospheric moisture, since CoPA is a hygroscopic material. The authors suspect that the crystalline behavior of CoPA, which has a crystallization temperature of 128 °C, is likely responsible for the upward warping of the overhangs, due to the observation during printing that this particular category of warping occurred while the deposited plastic was still well above its glass transition temperature. After constructing the heated chamber and drying the filament thoroughly before printing, print quality improved substantially, and with further tuning of print settings it was possible to achieve good print quality even for medium to high complexity objects such as the popular Benchy boat model [55]. The final major print settings used for CoPA (Polymaker), set in Prusa Slicer, are shown below in Table 3 alongside the print settings used for the deformation test objects for PLA (Polymaker) and ASA (Polymaker).

Table 3: Print Settings

Print Material	**Extrusion Temperature, °C**	**Bed Temperature, °C**	**Enclosure Temperature, °C**	**Layer Height, mm**	**Extrusion Width, mm**
PLA	210	60	20	0.2	0.4
ASA	260	105	20	0.2	0.4
CoPA	260	70	48	0.2	0.4

For large objects, especially objects with sharp corners, a large brim size was used for CoPA prints to improve build platform adhesion, or custom adhesion pads were added to the part design at sharp corners contacting the build platform.

The results of the deformation testing are shown in Figure 18. After autoclave, parts printed from both PLA and ASA exhibited clear thermal failure with severe warping and drooping, while parts printed from CoPA held their shape.

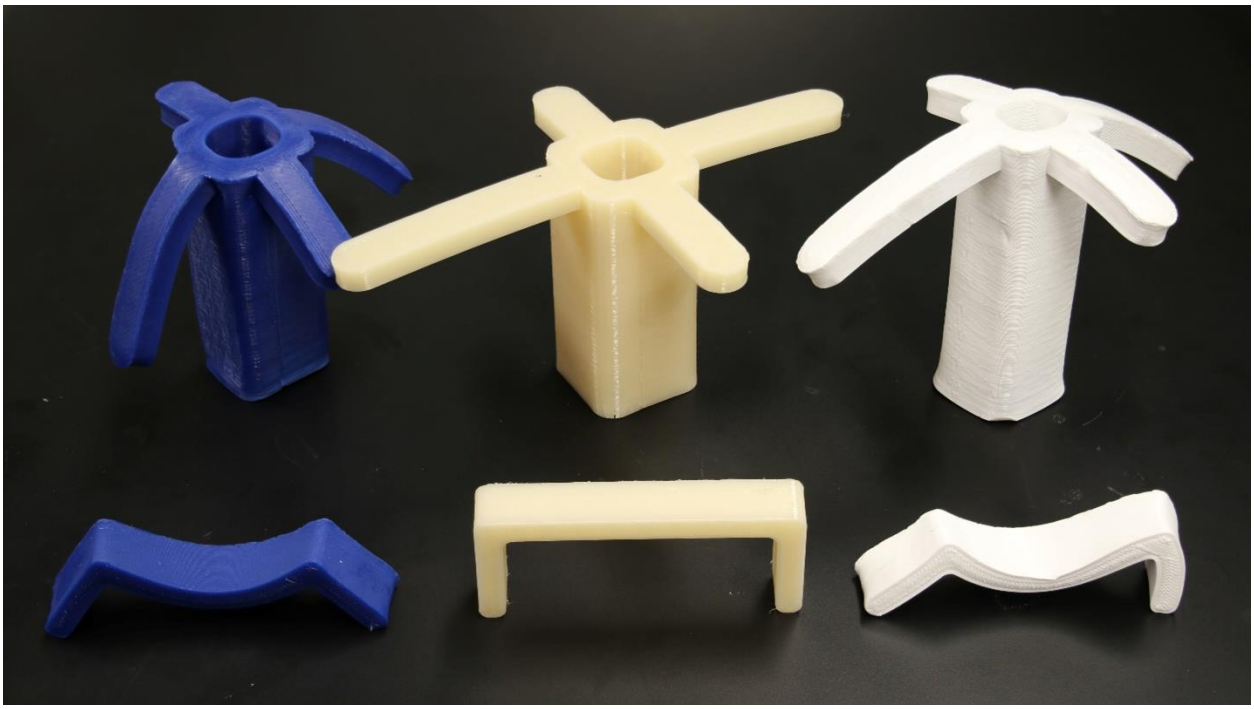

Figure 18: Post-autoclaving deformation testing results. Blue: PLA, Beige: CoPA, White: ASA. Top: Testing of cantilevers of various lengths. Bottom: Testing of bridges. PLA and ASA both fail to withstand autoclave temperatures for all cantilevers as well as the bridge, while CoPA has no detectable deformation on the bridge experiment and only minimal deformation on even the largest cantilevers tested.

Design and Printing for Uniaxial Tensile Test

Initially, tensile testing was difficult due to specimen failure at a weak point inherent in the 3D printed part where the extrudates spread outward as the specimen widens. The point where a central filling extrudate joined the spreading extrudates had lower strength than the rest of the specimen, and consistently failed first before any necking occurred in the specimen as shown below in Figure 3. Unsuccessful solution attempts included varying the infill pattern, re-melting the weak point area with a soldering iron, and packing salt around the prints and reflowing them in an oven. The weak point was eventually eliminated by processing the 3D model as 5 different pieces, separated by 1 micron, consisting of a central spar running the length of the specimen and 4 reinforcing wings which form the remaining width of the specimen grip. When using perimeter only infill by setting the number of

perimeters to be large as well as setting the part combination distance to be small compared to the piece separation, this results in a printed specimen with all 5 pieces fused with a consistent central spar with no inherent weak points running the full length of the specimen. Every extrudate in the testing area of the specimen runs parallel to the direction of tension, yielding accurate results of the material strength. A comparison of the toolpaths and resulting tested specimens is shown below in Figure 19 before and after manipulating the toolpath.

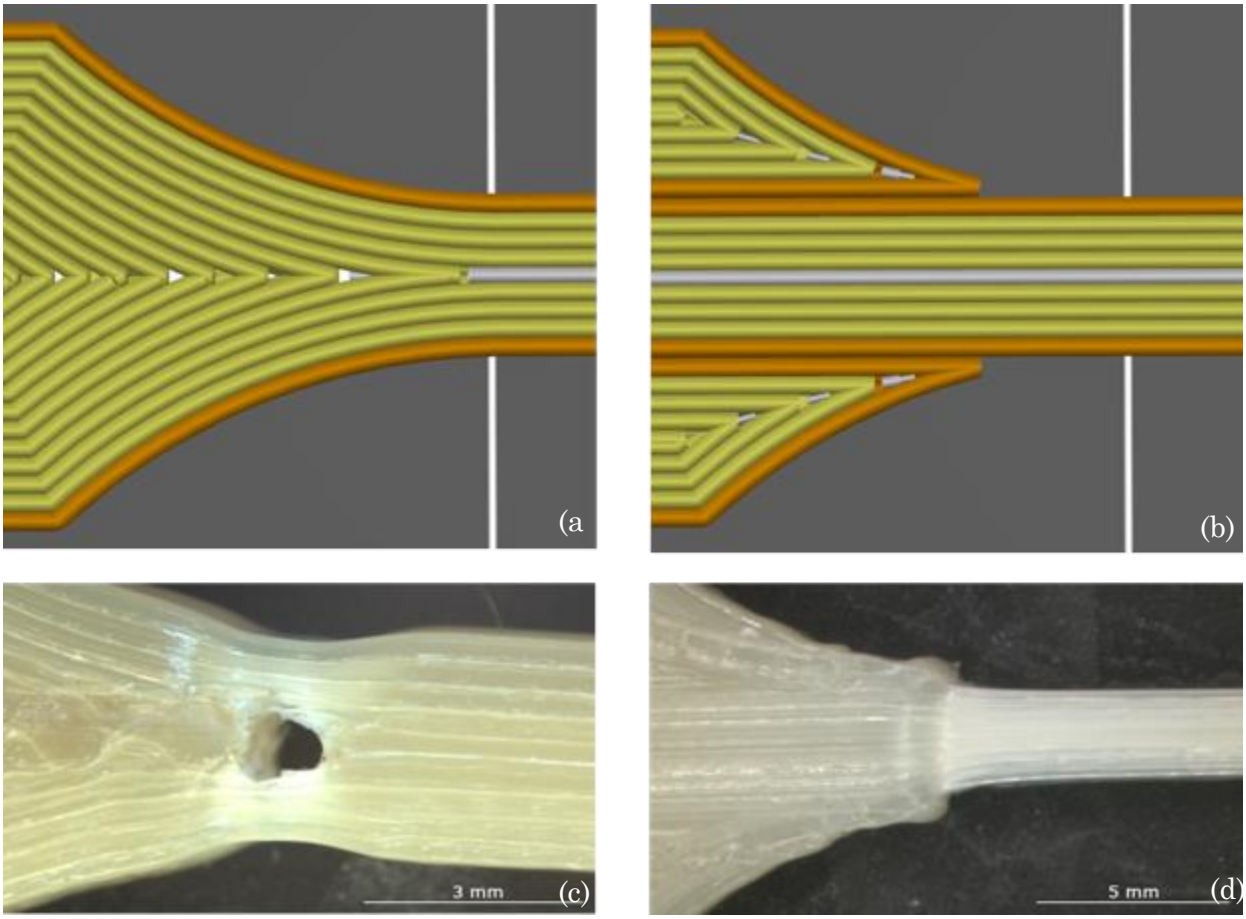

Figure 19: Toolpaths and tested specimens before and after manipulating the printer toolpath via part splitting and slight separation. (a) and (c): Before toolpath manipulation. (b) and (d): After toolpath manipulation via part splitting and separation.

Uniaxial Tensile Properties of 3D printed CoPA

The results of the tensile testing for mechanical degradation are shown in Table 4, reporting the mean engineering values of Ultimate Tensile Strength (UTS) and mechanical strain at UTS. Stress-strain curves showing the first 40% strain are also shown for qualitative comparison in Figure 20.

Table 4: Tensile testing results

	Mean UTS, MPa	UTS Standard Deviation, MPa	Mean Strain at UTS, %	Strain at UTS Standard Deviation, %
Non-Autoclaved, Wet	69.4	1.9	16.6	0.31
Autoclaved, Wet	65.8	0.7	17.1	0.75
Non-Autoclaved, Dry	73.4	0.9	17.8	0.32
Autoclaved, Dry	64.4	2.1	17.3	0.26

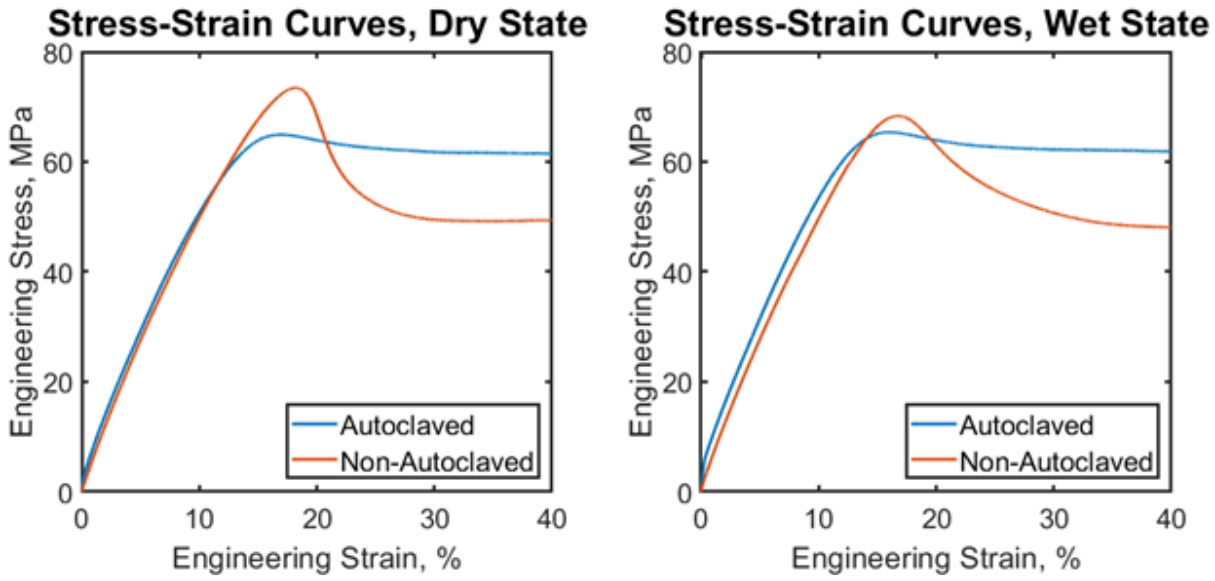

Figure 20: Representative tensile testing engineering stress-strain curves. Left: Samples after being dried in a sealed container with desiccant at reduced pressure. Right: Samples allowed to reach equilibrium with atmospheric moisture ('wet state'). Since specimens are 3D printed, there was variation between individual stress-strain curves, so the curves presented are the closest to a representative sample among the samples tested.

The table shows that the material withstands autoclave while maintaining strength and with little change to the engineering strain at Ultimate Tensile Strength (UTS). There

are several qualitative features of interest in the stress-strain curves shown above. Most notably, the autoclaved specimens on average have a slightly lower UTS, particularly for the dry specimens, but a substantially larger stress during the necking region of the curve. This increased necking stress is likely the result of annealing taking place during autoclave resulting in lower residual internal stresses and an increased resistance to slip in its semi-crystalline structure. The typical difference between dry and wet samples is also much lower for the autoclaved specimens, which suggests that the method of drying (2+ weeks in low pressure with desiccant packs) may be insufficient for removing the large amount of moisture collected during the autoclave process. Therefore, the authors recommend that future work use a vacuum oven to further reduce the pressure and lightly heat specimens to dry them more thoroughly if it is desired to study the dry state of the material. However, since the properties of the wet material lend itself well to face shield frames and other PPE as they are somewhat more flexible and therefore more comfortable, the authors do not recommend any drying of face shield frames after autoclaving and suggest that the frames go directly back into use after autoclave. It is also possible that the annealing caused by the autoclave temperatures somehow reduces the material's sensitivity to moisture content. More study of the basic material properties with respect to annealed and non-annealed states is suggested for future work. More study is also recommended on the number of times the material can be autoclaved without substantial degradation.

A face shield frame was printed with CoPA and subjected to a sterilization autoclave cycle. Only a small amount of deformation was noticed, and the part remained fully operational after the autoclave cycle, demonstrating that the material is suitable for manufacturing 3D printed autoclavable PPE. A face shield frame printed with CoPA is shown below in Figure 21.

Figure 21: A researcher wearing a face shield 3D printed from CoPA. After autoclaving face shields printed from CoPA there is only a small amount of deformation, and the part remains fully usable.

Conclusion

We presented a new material for 3D printing autoclavable personal protective equipment. We show that, unlike other commonly used materials such as PLA and ABS, CoPA, with its high Vicat softening temperature, can withstand autoclave temperatures without visible deformation and retains its mechanical properties post testing. Unlike other high-temperature polymers such as PEEK and PEI, which require industrial level 3D printers not typically accessible to mass consumers, CoPA with its relatively lower glass transition temperature can be printed with acceptable quality with only simple modifications to a commonly available consumer level 3D printer via a heated build chamber. Additionally, our uniaxial tensile experiments reveal that not only are ultimate tensile strength and elastic limit not substantially degraded when autoclaved, some properties, including some post-

necking hardening, are also moderately improved, likely attributable to the annealing effect during the autoclave process. We demonstrated 3D printing of face shields that remain functional after being subjected to autoclaving. This work points to a viable path to use widely available consumer 3D printers for autoclavable PPE manufacturing.