3D PRINTING PATH REALLOCATION FOR CONCURRENT IDEX SYSTEMS

An Undergraduate Research Scholars Thesis

by

SPENCER GAUTREAUX

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dylan Shell

May 2021

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Spencer Gautreaux, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

3D Printing Path Reallocation for Concurrent IDEX Systems

Spencer Gautreaux
Department of Computer Science
Texas A&M University

Research Faculty Advisor: Dr. Dylan Shell
Department of Computer Science
Texas A&M University

3D Printing is a growing field of interest, with research topics and commercial advancements in materials, processes, and systems. One of these advancements is the introduction of Independent Dual Extrusion (IDEX) Fused Deposition Modeling (FDM) printers in both the enterprise and consumer space. The unique feature on these printers is their dual extruders, which allows them to use multiple materials to create a printed part. These two extruders, in collaboration with two hotends, are responsible for the controlled deposition of material. In present systems, only one hotend can operate on the part at a time. However, as implied by the name, the hotends can be positioned independently. Therefore, the ability to utilize the two hotends concurrently could significantly reduce print time, a behavior not presently available. In this document we develop an algorithm to enable Collaborative Dual Extrusion (CODEX) printing, a model in which both hotends can be utilized simultaneously on one part. To do so we outline a two-phase greedy algorithm for transforming an input GCode file, intended for a traditional FDM printer, into one that could be utilized on IDEX printers. This algorithm exploits the sequential nature of GCode to find large runs of concurrently printable segments. These runs are then linked to produce output paths. Approximately 13,500 publicly available GCode files are utilized to test and validate the algorithm across three different conceptual models for IDEX printers. The first model provides a theoretical maximum upper bound on efficiency. The second represents a mechanically feasible model. The final model simulates those IDEX printers available today. We show an approximate 24% and 20% improvement for the first two models, and a 9% deterioration on the final model. The document concludes with a discussion of possible improvements and directions for future work.

DEDICATION

To our friends, families, instructors, and peers who supported us throughout the research process.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. Shell, for his guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience, especially to the group that put this idea in my head a year ago.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

We received no outside funding for this work.

NOMENCLATURE

Base Efficiency	The ratio of raw base time to base time
Base Time	The time, including the non-print movements in the recomputed file
Chain	A collection of consecutive, coplanar segments
Chain-Pair	A pair of two chains, one for each hotend
Minimum Separation Distance	The minimum distance between two hotends without interference
FDM	Fused Deposition Modeling, a standard form of 3D printing
GCode	Standard file format for Computerized Numerical Control (CNC) machines, used by most consumer 3D printers
G1	An instruction in GCode representing a movement of linear interpolation from the current position to the position specified by the instruction
Hotend	The component of an FDM 3D printer responsible for heating and depositing material to realize the printing process
IDEX	Independent Dual Extrusion, a form of 3D printing printer
Layer	A collection of segments that are all coplanar
Position	A (X, Y) coordinate point in a specific Z-Layer
Print Efficiency	The ratio of raw print time to print time
Print Segment	A segment which contains an instruction in the print dimension, resulting in the deposition of new material
Print Time	The time of all print segments in the recomputed file
Raw Base Time	The base time of the original file
Raw Print Time	The print time of the original file

Recompute	The process of rearranging the file in order to utilize both hotends concurrently.
Segment	A single, linear move from one position to another position, represented by G1 in GCode
State	A pair of two positions, corresponding to the positions of the two hotends
Z-Layer	A layer where the plane shared by all segments is parallel to the plane $Z = 0$

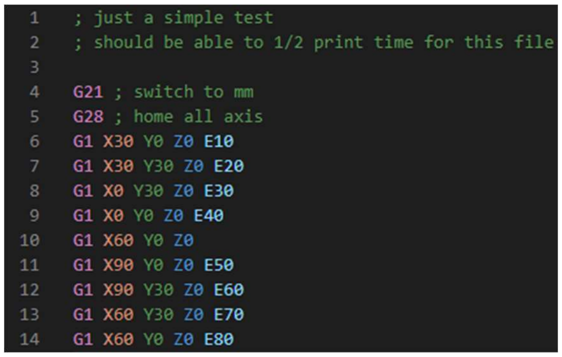
1. INTRODUCTION

1.1 Fused Deposition Modeling Process Introduction

Fused Deposition Modeling (FDM) is one of many forms of 3D printing [1] [8] [10]. In this printing format, consecutive layers of material are deposited to construct a larger part. On each layer, a hotend is utilized to control the deposition of material, traditionally a thermoplastic, in a controlled pattern. By stacking several of these layers, a 3D printed part is constructed.

GCode is a semi-standardized file format for Computerized Numerical Control (CNC) machines. The GCode is generated by a program known as a slicer which translates a 3D model into GCode [8] [10]. This process involves many parameters for controlling the FDM machine. Selection of these parameters are non-trivial as the parameters greatly influences the output part [1] [8]. However, by operating at the GCode layer, much of this complexity can be ignored as it is encoded into the path within the GCode file.

Each line of the GCode file contains a command for the machine. The most prevalent command in the 3D printing space is G1: linear interpolation. The G1 command specifies an end point to which the machine must move. This forms a segment, with the start point being the machine's position prior to the instruction being ran. Any axis not specified, is simply left unchanged. Examples of additional axis are 'E' for extrusion and 'F' for feed rate. A more through discussion of the GCode file format can be found on the RepRap glossary [12]. Figure 1.1 shows example GCode and figure 1.2 shows the path corresponding to the GCode in Figure 1.1.



```
 1  ; just a simple test
 2  ; should be able to 1/2 print time for this file
 3
 4  G21 ; switch to mm
 5  G28 ; home all axis
 6  G1 X30 Y0 Z0 E10
 7  G1 X30 Y30 Z0 E20
 8  G1 X0 Y30 Z0 E30
 9  G1 X0 Y0 Z0 E40
10  G1 X60 Y0 Z0
11  G1 X90 Y0 Z0 E50
12  G1 X90 Y30 Z0 E60
13  G1 X60 Y30 Z0 E70
14  G1 X60 Y0 Z0 E80
```

Figure 1.1: Sample GCode as viewed in Visual Studio Code with the "nc-gcode" syntax highlighter [6] [7].

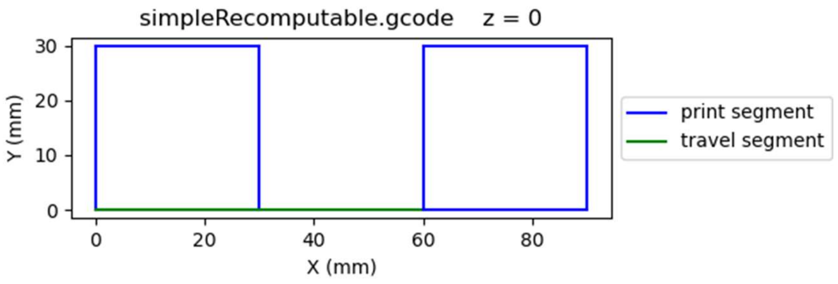


Figure 1.2: Path represented by the GCode in Figure 1.1. Note, the print segment from (0,0) to (30,0) on line 1 is occluded by the latter movement segment from (0,0) to (60,0) on line 10. Figure generated using Python3 matplotlib [5].

1.2 Independent Dual Extrusion Introduction

Independent Dual Extrusion (IDEX) is an evolution of the FDM process. Rather than a single hotend, and IDEX printer contains two hotends. These two hotends can be employed in a variety of ways. In one scenario, the two hotends are used collaboratively on the same part. In this scenario, the two hotends contain different materials, allowing the resulting printed part to selectively employ the differing material properties to its benefit. In the simplest case, this is utilizing two colors to create a final part with some artistic design. Another scenario for IDEX printers involves the use the two hotends to print two copies (or mirrored copies) of the same part. These features are already available in commercial IDEX 3D printers [3] [11].

However, one key feature is missing from all IDEX printers: the ability for the two hotends to work together on a single part. In theory (and as shown later in practice) these two

IDEX heads can be employed collaboratively to reduce the print time of a single part, allowing an increase in the production speed of the part. This collaboration is subject to some physical limitations of the printer. In the most common IDEX format, the two hotends are linked along a single gantry. While the hotends are independent in one axis, for example the X axis, they are fixed relative to each other in the Y and Z dimensions. This is a tightly constrained system and thus will offer less opportunity to optimize. Therefore, a more relaxed format can be envisioned where each hotend is on its own gantry [14]. Thus, the hotends would be independent in X and Y. While this is not a commercially available system today, it represents what IDEX printers may look like in the future.

1.3 Prior Research

FDM is an active area of research [1] [8] [10]. As such, the path planning aspects and GCode generation can be considered to also be active. Unfortunately, there is less research within the IDEX space. However, some benefits from FDM research will impact IDEX printers. For example, the two hotends in an IDEX system can be argued to each be functionally equivalent to a standard FDM printer. Thus, any general improvement to FDM printers should also improve IDEX printers.

The path planning problem addressed in this research is one of significant complexity, complexity stemming from the need to coordinate multiple agents while respecting their geometric constraints. Fundamentally, this is an offline planning problem, one that could be solved in any number of ways [13]. However, this planning problem differs in that a solution is already provided. The input files, GCode files for traditional FDM printers, already contain a pathed solution for a single agent. Therefore, this problem becomes one of converting a single-agent solution into a multi-agent one.

2. METHODS

2.1 The Layer Recompute Process

The entire recompute process operates on a layer-by-layer basis. Within each layer, a two-phase greedy algorithm is applied [15]. The first round focuses on finding portions of the layer which can be concurrently printed by the two hotends. The second round focuses on linking the outputs of the first round into a continuous path. Three unique recompute models are considered, representing three types of physical constraints which are introduced by the underlying construction of the machine. These constraints and their motivation are discussed in the IDEX Configurations Considered section. The recompute process also makes use of a constant, the minimum separation distance, to determine which segments can be printed without collision.

Within the recompute process, the fundamental unit is the segment. Segments are considered in pairs, looking for pairings which satisfy the question: “Can hotend H1 print segment S1 while hotend H2 prints segment S2, without colliding in any configuration?” That is, there exists no positions P1 on S1 and P2 on S2 where P1 and P2 are colliding. Here “colliding” is taken to mean either the hotends being within the minimum separation distance, representing a physical collision, or the hotends being in a position inconsistent with the underlying physical system, an impossible configuration. Where the segments can be printed concurrently, it implies that the two hotends can print their segments without collision or synchronization. If both hotends follow a continuous path of segments throughout the file and synchronize at each end point, then the recomputed paths can be assumed to be collision free in the three-dimensional X-

Y-Time space [9]. In this model, the Z dimension can be ignored as all print segments exist in a Z plane and planes are printed in strictly increasing order.

The segments originate from prepared GCode files. The G1 (liner interpolation) commands are extracted and converted to a corresponding segment. Only the segments that correspond to a print are kept for the recompute process, the non-print segments are discarded. If the segment exceeds the minimum separation distance, it is split into several sub segments each at most one minimum separation distance in length. This is done to increase the resolution of the collision checking. For example, two long segments which share an endpoint may be considered colliding by the prior definition. However, when split into smaller segments, many of these segments are non-colliding and may be concurrently printed. Secondly, this splitting guarantees that all the segments are relatively equal in length. This allows for some simplifications such as treating the time to print a segment as a constant. More importantly, this prevents the need for one hotend to idle while the other prints a long segment.

The GCode file implies a temporal ordering of segments. This is maintained through the splitting phase. If a segment is split, it is split starting at the start point and every minimum separation distance until the entirety of the original segment is covered by these new segments. The final segment may be shorter than a minimum separation distance. This splitting process is applied only to print segments and the new segments are inserted in temporal ordering.

The problem then becomes efficiently finding groups of segments that can be printed concurrently. To do this a structure called a *chain* is employed. A chain is a collection of segments that are printed consecutively in the GCode file. These chains have several unique properties that make them extremely interesting. First, the chain definition ensures that end point of segment S_n is also the start point of segment S_{n+1}; these two segments can be printed without a traversal in-between them. Secondly a chain implies locality. Since these segments are connected via their endpoints there is an implied geometric locality. If two segments (in two chains) are in a valid position, there is a high likelihood that the pair of the next segments (in the chains) are also concurrently printable. Thirdly, both of these properties also hold on the chain if iterated in reverse. And finally, a chain is itself composed of many sub-chains. A single segment is a chain, or the entire layer can be a chain.

The recompute process consists of two phases of a greedy algorithm. The first phase seeks to construct chain-pairs such that all the segments are present in a single chain which is itself in a single chain pair. The chain pairs are constrained such that the chains are of equal length. Thus, when the chains finish, they are both ready to transition to new positions to begin the chain process again. It is phase two that manages the computation of these transitions.

This process is repeated for each layer until the entire file has been printed.

2.2 Phase One Recompute

Phase one seeks to construct chain pairs, where each hotend prints one of the chains. The chains in each pair should be equal in length. The lengths of pairs should be maximized, in order to minimize the work in phase two. Unfortunately, not all chains can be considered; there are simply too many. Each segment can be a chain. As well as each pairing of two consecutive segments. As well as each tuple of three consecutive segments… Then each chain would need to be considered in the forward and reverse direction, and then against all other chains. This quickly grows to an untenable number. Rather only the chains as they are expressed in the GCode file are considered. That is, the chains of maximum length incident on each end with a non-print segment. These chains are considered in both the forward (as represented in GCode) and reverse directions. This does an excellent job in reducing the number of chains.

The chains are considered against the other chains in this layer. For each possible pairing the chains are traversed to determine the maximum number of segments that can be printed concurrently, terminating when the segments collide or one of the chains is exhausted. The longest chain-pairing is set aside for phase two. Either (or both) chain(s) may produce a remainder chain which must be considered, in the forward and reverse direction, against all other chains. This process repeats until all segments are in a chain pair.

However, the number of chains is often still too large to compute the best pairing out of all the pairs possible. In this case, the number of concurrent chains under consideration can be limited to some value. For the experiments in this paper, the threshold was set to 100 chains in consideration. All the chains begin in a pending set ordered on chain length. The considered set is initialized by promoting chains from the pending set, up to the limit. The best pairing of chains in the set is found, the pairing set aside for phase two, and any remainder chains computed. The remainder chains are then reinserted directly into the considered set, bypassing the pending set. This was no motivation for this behavior versus returning the remainders to the pending set. If either (or both) chain(s) did not produce a remainder, the considered set can be backfilled from the pending set. The process repeats until the pending set and considered set are both empty. If this process reaches a deadlock wherein the considered set is unable to find any valid pairings, the considered set can be temporarily expanded by taking elements from the pending set and repeating the process. This appears like it could quickly expand the number of pairs in consideration and thus significantly increase runtime. However, in practice this is not an issue. For the experiments in this paper, when this deadlock condition was reached, the next five largest chains were moved from the pending set to the considered set.

It is of course possible, and likely, to encounter a scenario with a few short chains of one to two segments in length that are all mutually colliding. These remaining chains are then placed into pairs with special empty chains to ensure that all segments are present in a chain pair.

Consider a chain that forms a loop where the start and end point of the chain are coincident. Since the chains are evaluated from the outside in, their start and end segments are colliding. Thus, the chain pair of the forward and backward iteration is considered invalid. However, it is possible to have chains in the middle that could be concurrently printed if the chain were split. These scenarios are missed by this approach and no attempt is made to recognize such configurations.

2.3 Phase Two Recompute

In comparison, the phase two recompute is much simpler. It takes the chain pairs generated in phase one and links them together in a logical manner. Each chain pairing specifies two states. Each state consists of a pair of (X, Y) positions, one for each hotend. Notice that by construction every state must be valid: it represents two endpoints of a valid chain pairing. Therefore, for two sates each in a valid configuration of the underlying printer, a transition between states should be possible. Further, notice each state is unique. If two states are the same, then the incident segments are valid and consecutive, and would have resulted in a longer chain pair in phase one.

The transition time from one state to another can be calculated according to the following method. For each hotend consider the time to transition linearly from one position to the other, this defines a transition segment for the hotend. The transitions time is the length of this segment divided by minimum separation distance and rounded up to ensure an overestimate is used. If both hotends can move their segments concurrently the transition time is the longer of the two segments. If this check fails, check if the hotends can be moved one segment then the other. In this case the transition time is the sum of the two movement segments.

If the transitions cannot be printed concurrently or consecutively, the transition time is taken to be the sum of the two segments, rounded up, plus some delta. For this thesis, the delta was four times the minimum separation distance. This is designed to represent the scenario in which one hotend will move one minimum separation distance off its position, the other hotend will traverse its segment, and then the hotend will return to its original position. This process is then repeated for the other hotend. Of course, this is a significant overestimate: it can be improved by not returning to the original position and rather traversing to the ending position directly. However, using the delta above represents an absolute worst-case scenario.

If one of the chains in the chain pair is an empty chain, the transition time for that hotend is zero.

It is in phase two that the reversibility of chains is paramount. Recall, that a chain can be reversed to allow for traversal in the opposite direction. Thus, a chain pair, being two chains of the same length, can also be reversed without needing to recompute the collisions. Therefore, the end position of a chain pair can link to the end position of another chain pair though an implied reversal of one of the chains. This new chain pair-transition-chain pair structure itself maintains the property of reversibility.

Therefore, linking chain pairs is as simple as linking states. This process is similar to Kruskal’s algorithm for minimum spanning trees [15]. A series of groups, one per chain pair, is constructed, containing the two states incident on the chain pair. Simply find the two closest states that belong to different groups, add this transition, join the groups, and repeat until only one group remains. Phase two concludes once all the chain pairs are linked. At this point, a valid path has been constructed for the layer, and the recompute process can begin on the next layer.

2.4 Layer Recompute Assumptions

Some fundamental assumptions are made on the input files to aid in the recompute. In aggregate, these assumptions relax the constraints on the printing system, allowing for a solution to be found more easily. These are not inherently necessary to the algorithm, and, as is covered in the section Tightening Constraints on the Solutions, are largely arbitrary at this point. The assumptions are that segments within a single layer are freely reorderable, reversable, and distributable. Reorderable means that any two segments, A and B, in a layer can be printed A before B, B before A, or A concurrent with B, without concern for the order these segments appear in the original file. Reversable means that the segment can be printed from the start point to end point as expressed in the GCode file or from the end point to the start point. Finally, the segments are assumed to be freely distributable, that either hotend can be responsible for printing the segment.

At all times, the endpoints of segments, and thus the segments themselves, remain unchanged. However, the assumption is made that some segment A can be split into two or more subsegments for which all of the earlier properties hold.

Another key assumption made is that the printer mechanically is sufficiently large to print the file: that in some configuration of the mechanical components, either hotend can print all the segments in the file.

2.5 IDEX Configurations Considered

Three models of IDEX printer are considered: a Theoretical Model focused on absolute efficiency, a CODEX Model based on a mechanically feasible printer, and a Current Model which is focused on currently available IDEX machines.

All three models require some supporting functions, which are defined in the following sections. These functions answer three fundamental questions:

1. Can hotend H1 hold some position P1 while hotend H2 holds some other position P2?
2. Can hotend H1/H2 move some segment S1/S2 while hotend H2/H1 holds some position P2/P1?
3. Can hotend H1 move segment S1 while hotend H2 moves segment S2?

The answers to these questions are employed in the recompute process and represent the only change between the various models. Note the condition 2 is not necessarily symmetric, for the same segment-position pair, it may be valid for hotend H1 to hold the position while hotend H2 traverses the segment, while the reverse may not be true.

In the Theoretical Model, the hotends are assumed to be freely positioned in 3D space with little regard to the physical components required to realize that position. While this configuration does not represent the physical behaviors of current IDEX printers, it provides an upper bound on the approximate efficiency achievable in IDEX printing. Both of the following models represent a more constrained system. Therefore, any solution for those models must be, in theory, no better than for this model. Of course, due to the greedy nature of the recompute algorithm this behavior is not always realized.

The next model, the CODEX Model, proposed by Reddit user u/m47812 [14]. In this model the hotends are fixed on gantries, limiting their range of motion by preventing crossing in the X axis. This model is not presently commercially available; it requires a certain mechanical complication that is pointless without the path planning ability to utilize the print heads concurrently. Therefore, while this model is not truly real-world, it represents a model of a printer that could be developed.

Finally, the Current Model explores how current commercially available models are able to implement the collaborative behavior without physical hardware changes. In essence, this is the "today metric", what could be achieved right now with a simple over-the-air update. It is the most constrained, with the two hotends sitting on a single gantry. This means that the hotends are linked in the X axis and must not cross (or interfere) in the Y axis.

The following figure, Figure 2.1, shows a top-down representation of the printing models. The hotends are represented by circles. In the theoretical model, these hotends move freely throughout the X-Y space. In the other two models, the motion of the hotends is limited by mechanical gantries, represented by the gray bars. These gantries are responsible for the additional constraints placed on the CODEX Model and Current Model.

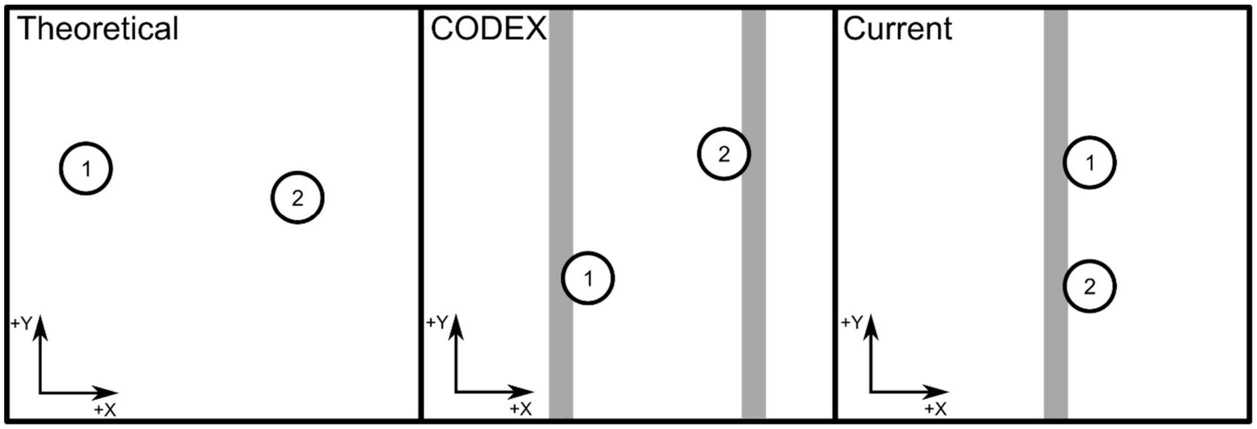


Figure 2.1: Image of the three models considered. The circles indicate the hotends while the gray bar indicates the gantries to which the hotends are attached and cannot pass. Gantries can move only in the X dimensions and extend infinitely in the Y dimension.

2.6 Formalizing Constraints on the Theoretical Model

In the Theoretical Model, the hotends must maintain at least a 25 mm separation between points. This should hold true for the distance from a position to any position on segment or between any two positions on two distinct segments.

The 25-millimeter distance is taken from the model of E3D V6 hotend; 25 millimeters is roughly how close together two of these hotends can get without colliding [4]. Additionally, the E3D V6 was chosen as it is a round hotend, making the collision calculations easier.

2.7 Formalizing Constraints on the CODEX Model

In the Codex Model, the hotends must maintain at least 25 mm separation, the same as in the Theoretical Model. Additionally, for all possible position pairings, hotend H1's X coordinate must be less than or equal to that of hotend H2. That is, hotend H1 cannot cross the X coordinate of hotend H2.

2.8 Formalizing Constraints on the Current Model

The Current Model is somewhat more complex. A trivial solution is to take the constraints exactly as they appear in the real world: that X coordinates must be equal and that hotend H1 must be at least one separation distance (again 25 millimeters) greater in the Y dimension than that of the hotend H2. Unsurprisingly, this will result in almost no segments being able to print concurrently. Therefore, the constraints must be somewhat relaxed. This is accomplished by allowing positions/segments to have some amount of X deviation. There are two interpretations of this behavior.

First, consider the gantry as moving in X as a function of time T. Then, the segments can be shifted in time so that their start and end points line up in X but not in T. Since the segment chains are continuous, this is a valid transformation. However, it potentially violates non-collision property with other segments. In essence, this would require checking the segment S1 (printed by hotend H1) with both of the overlapping segments printed by the other hotend (H2), and ensuring the non-collision is maintained. This is ideal as it would not increase the print time (aside from some lead in/out at either end of the chain pairs where only one hotend can be printed).

In the other interpretation, once one of the hotends finishes printing its segment, it can simply remain in place in the Y axis until the other hotend prints its segment. Then the machine can back-track to the end point of the segment assigned to the first hotend, allowing it to begin printing the next segments, enabling the second hotend when the proper X coordinate is reached. This is not ideal as it increases the print time by up to twice the ratio of X deviation to segment length.

The physical and temporal behaviors of these scenarios are not accounted for in the model. That is, it is assumed that phase shifting is always a valid solution to this problem and the lead in/out time is ignored. The maximum allowable X deviation was set to 5 millimeters.

2.9 The GCode Dataset

The GCode dataset prepared by Baumann provides an excellent starting point for this research [2]. It provides a series of real-world models, publicly available, and intended for 3D Printing. Furthermore, this dataset contains several pre-generated the GCode for these models, providing a significant collection of real-world models to validate these algorithms. The files are labeled numerically and distributed in shards based on the first number. The dataset files for 2*-9* were downloaded. However, the 1*-set of files repeatedly failed to download. Next, the downloaded files were limited to only those that were less than 32 MB in size to facilitate the multi-processing of the data files without overloading the available computing resources. This results in a total of 13,591 GCode files from the dataset.

All of the selected files are in the Z up configuration, meaning layers run in a single Z plane. Additionally, all the files are monotonically increasing in Z. All the files are processed without error, indicating that they are validly constructed GCode files. The files contain only G1 (linear interpolation), G28 (axis home), and M (non-move control) codes. For all the files, all the print segments are in a singular Z-Layer. In summary, all 13,591 files of the reduced dataset are valid GCode files.

2.10 Data Flow

The GCode files are parsed to extract the segments, as specified by the G1 commands. The segments are then pre-processed; if the segment exceeds the minimum collision distance, the segment is split. After the segments are read in, a series of secondary check are run on the file. The file is checked for monotonic increasing segments and other error conditions.

Next, some metadata is collected on the base file: the Raw Print Time, the total time spent printing in the original file, and the Raw Base Time, the total time spent printing and moving in the original file. The Raw Base Time includes intra-layer non-print segments but does not include the inter-layer non-print segments.

Next, the layer-by-layer recompute process is performed. Each layer reports its Print Time and Base Time. This is collected into an overall metric for the entire file. Then the program computes the two-primary metrics. First, the program computes the Base Efficiency, the ratio of Raw Base Time to Base Time. This gives a ratio of overall performance improvement and is optimally slightly greater than 2.0. This occurs where two hotends can work at 100% efficiency and with no transition segments in the recomputed path. With intentionally designed input files, the Base Efficiency could be pushed to positive infinity. This is done by providing an input with two print segments connected by an infinite number of non-segments. However, this is not representative of the real-world inputs used in these experiments. The second metric is the Print Efficiency, the ratio of Raw Print Time to Print Time. This ratio is locked in the range 1.0 to 2.0; values outside this range would represent the loss or addition of segments in the new output file. Together these metrics form the basis for analyzing the algorithm's performance.

2.11 Logistics of the Experiment

A Python3 script was developed that managed the recompute process detailed above. For each input GCode file, the script launched a new process via the built in SubProcess module to recompute the file. The script spawned up to sixteen concurrent subprocesses. Sixteen was chosen as this was the hardware thread count supported by the CPU, allowing for one file to be processed per CPU core. Each subprocess logged its output to a file reporting per-layer and per-file output metrics. The output of the script logged the files for which processing exited abnormally, as well as the overall wall time of the batch process. The script was run three times, once per recompute model.

Several secondary Python3 scripts were developed for post processing of the output. These scripts collected and correlated key output metrics from the per-file and overall output files, constructed CSV files, performed data analytics, and generated the charts and graphs in matplotlib.

2.12 Hardware Utilized for Computations

The code for this project was written in C++ 2017, compiled with g++ 8.4.0 and run with Windows Subsystem for Linux 2. The underlying hardware was an Intel i7-10700K CPU which was slightly overclocked. The Windows™ build utilized was 19042.906.

3. RESULTS

3.1 Metadata on the Experiment

Some key metrics are summarized in the table below. Each processing batch contains exactly 347 error files. These 347 files are the same across all three models. Additionally, the per-file outputs are consistent with these error counts. These files are the ones which exceed 32 MB in size and were excluded for their large size. No other files reported errors.

One interesting result is the processing time. The model utilized significantly impacts the overall runtime. The recompute runtime is greatest with the Theoretical Model. Because the solution is less constrained, more time is spent in phase one matching chain pairs. By contrast, in the Current Model, very few chain pairings are valid. Therefore, the recompute algorithm quickly exits phase one on each layer, saving significant time. Table 3.1 quantifies these metrics. Since the recompute likely occurs before printing occurs, the runtime of the recompute is unimportant. However, it is interesting to note that the Current Model runs, in aggregate across all files, almost five times faster than the Theoretical Model.

Table 3.1: Comparisons of basic output information for the three models considered.

	Theoretical Model	CODEX Model	Current Model
Wall Time Taken (s)	13,756	9,574	6,045
CPU Time Taken (s)	163,405	95,345	30,215
Qty Error Files	347	347	347
Longest Recompute Time (s)	523	296	207

3.2 Specific Metrics on the Results

For each of the three recompute models, several statistics are shown, in the following table, Table 3.2. There are two principal metrics under consideration for each file. First, is Print Efficiency. This is the ratio of time spent printing in the original file as compared to the time spent printing in the recomputed file. The second metric is Base Efficiency, the ratio of total time, including non-print time, in the original file, as compared to the total time in the recomputed file. In essence, the first is a measure of the ability to locate and match segments for concurrent printing while the latter measures the overall efficiency of the process. Additionally, the standard deviation for each model and each statistic is shown.

Table 3.2: Comparisons of key output metrics for each model. All numbers are over the entire dataset.

	Theoretical Model	CODEX Model	Current Model
Average Print Efficiency	1.55498	1.54646	1.13257
Average Base Efficiency	1.24340	1.20960	0.91698
Standard Deviation Print Efficiency	0.39201	0.38945	0.19160
Standard Deviation Base Efficiency	0.31105	0.28907	0.11001

Unsurprisingly, the Theoretical Model has the highest efficiency. This is expected as it is less constrained than either of the other two models; any solution for those models would also be valid for the Theoretical Model. Surprisingly, the CODEX Model is nearly as performant as the Theoretical Model. Both exhibit print efficiencies of nearly 1.55, a roughly 55% improvement in time spent printing. While there is significant improvement in the print time, the recomputed path also comes with significant traversal overhead. This results in an increase in total time and a reduction in Base Efficiency. This is most noticeable in the Current Model, wherein the amount of non-print traversal time causes the Base Efficiency to go below 1.0. When the Base Efficiency is below 1.0, the files now take longer to print. This is vexing, but this statistic is somewhat deceptive. There are many files with very low Raw Print Times that optimize poorly. For example, the following figures show comparisons of Print Efficiency to Raw Print Time and Base Efficiency to base print time.

The following two figures help to further contextualize these results. Figure 3.1 shows three scatter plots, one per recompute model. Each plot shows Print Efficiency in relation to Raw Print Time. Each point is one file from the dataset. Therefore, Figure 3.1 effectively shows the improvement for each file with respect to its original print size. Following Figure 3.1, Figure 3.2 shows a histogram of Print Efficiency for all files for each of the models.

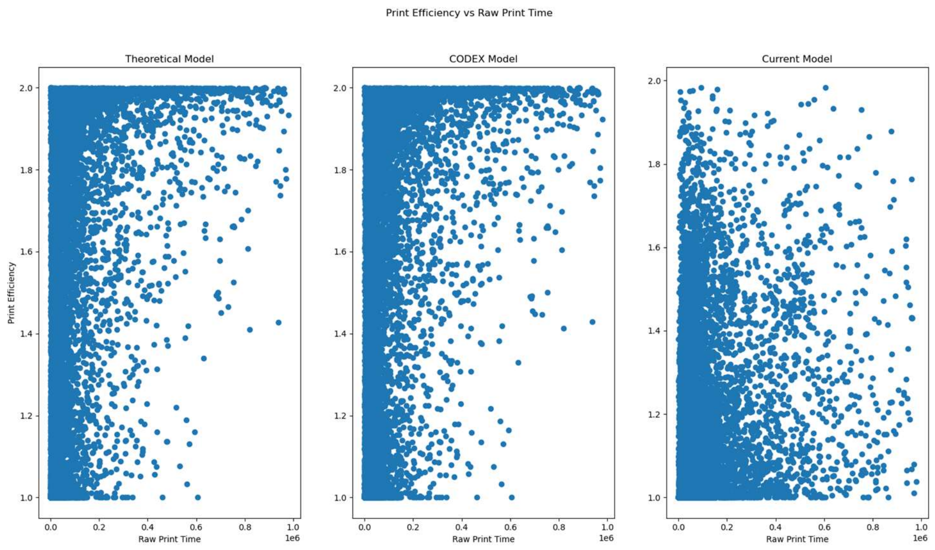


Figure 3.1: Scatter plots comparing Print Efficiency to Raw Print Time.

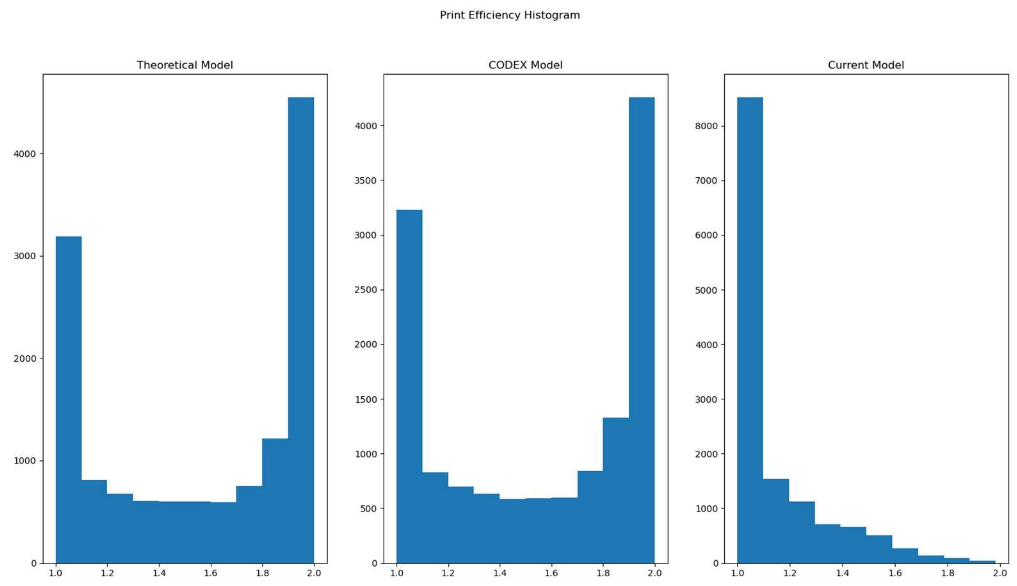


Figure 3.2: Histogram of Print Efficiency per model. Notice the Y-Axis scale differs between models.

The results for Print Efficiency are extremely encouraging. For both the Theoretical Model and the CODEX Model, a majority of files can experience significant reduction in overall print time. This is an excellent outcome. It can be seen in the scatter plot and the histogram that the Print Efficiency is bounded between 1.0 and 2.0. This follows the expected behavior; a value less than 1.0 would indicate segments being added in the recomputed file while a value over 2.0 would indicate segments being lost. Figure 3.1 shows a weak correlation between original file size and Print Efficiency in the Theoretical Model and the CODEX Model. This makes sense as a larger file can recomputed more easily. Unfortunately, this relationship does not exist in the Current Model. The histograms for the Theoretical Model and the CODEX Model both have large grouping at the end of the 2.0 spectrum, which indicates many files can have significant reductions in time spent printing. However, in the Current Model, much of the sample data is grouped to the 1.0 end of the spectrum. This is an undesirable result indicating that often little improvement is possible. However, for those files at the 1.0 end, approximately half of them cannot be improved by the Theoretical Model. If the Theoretical Model cannot improve the file, the Current Model will of course do no better. Nevertheless, the Current Model is certainly less performant than its counterparts.

Next follows two additional figures. Figure 3.3 is another scatterplot, similar in construction to that of Figure 3.1. The primary difference is that Figure 3.3 plots Base Efficiency as a function of Raw Base Time. Additionally, all items of less than 1.0 Base Efficiency are colored in red and demarked with a plus symbol. These are the items for which the recompute produced an overall worse result as compared to the raw file. Figure 3.4 is a histogram of Base Efficiency. Again, the regions of less than 1.0 Base Efficiency are colored in red.

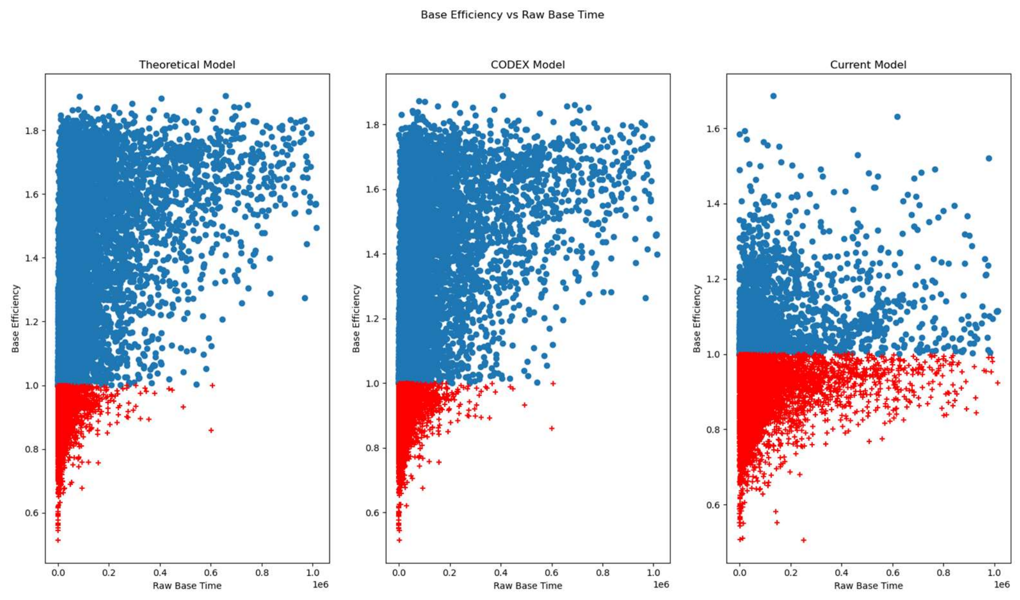


Figure 3.3: Scatter plots comparing Base Efficiency to Raw Base Time. Files in red, marked with '+' are below 1.0 efficiency. Notice the Y-Axis scale differs between models.

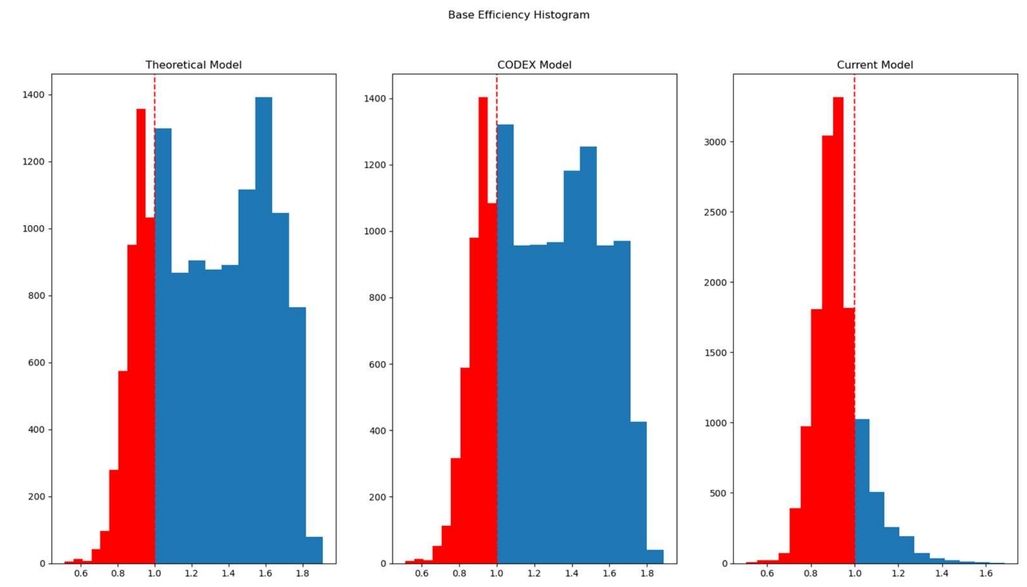


Figure 3.4: Histogram of Base Efficiency per model. Regions in red indicate files below 1.0 efficiency. Notice the X-Axis and Y-Axis scale differs between models.

The performance of the algorithm gets worse when looking at the Base Efficiency. Many files have below 1.0 Base Efficiency. This means that the recomputed file takes longer than the original file. Specifically, only 67.9%, 66.4%, and 15.6% of the files show any improvement for the Theoretical Model, the CODEX Model, and the Current Model, respectively. This is irksome.

3.3 Secondary Metrics on the Results

As seen in the scatter plot Figure 3.1 and Figure 3.3, there is a bias towards smaller files in the dataset, and smaller files tend to be less optimizable. Therefore, it is natural to consider the weighted average of the Print Efficiency, weighted on the Raw Print Time, and of Base Efficiency, weighted on Raw Base Time. The results of this calculation are shown in Table 3.3. To some extent, this paints a better picture. Both efficiency metrics improve significantly. However, the Current Model is still beneath 1.0 Base Efficiency on average.

Table 3.3: Comparisons of additional output metrics for each model.

	Theoretical Model	CODEX Model	Current Model
Weighted Average Print Efficiency	1.77317	1.76789	1.23150
Weighted Average Base Efficiency	1.46642	1.42262	0.97980
Max Print Efficiency	2.0	2.0	1.98258
Max Base Efficiency	1.90786	1.88841	1.68580

Another interesting statistic appears in considering the most performant files. Both the Theoretical Model and the CODEX Model have recomputed files with Print Efficiency at the absolute maximum 2.0. Even the current model comes close at a 1.98. The maximum Base Efficiency is equally as impressive, suggesting a 90% and 88% improvement in the best case for the Theoretical Model and the CODEX Model respectively. They do not achieve the theoretical maximum Base Efficiency, which is greater than 2.0, however these results are quite promising. Even the Current Model performs well, producing a 69% improvement in the best case. In summary, all three models can produce incredibly performant results, given the right input file.

4. CONCLUSION

4.1 Discussion of Results

Overall, the results are quite encouraging. With respect to Print Efficiency, the results are excellent. The algorithm presented can successfully reallocate the segments to significantly reduce time printing. However, in doing so too much non-print transition time is introduced, frequently making the solution worse. This is likely an artifact of the recompute process: it maximizes concurrent printing first with little regard for the transition behavior. Despite this behavior, the recompute process can create significant improvements in all three models which is promising for future work. Furthermore, the entire second phase has room for improvement. Principally, this is in the estimation of transition time, which is currently overestimated in most cases. However, a larger problem arises in the fact that the printer is clearly underutilized. Right now, the printer is either printing two segments concurrently or moving to do so. A key omitted behavior is in printing while another hotend repositions. This could allow for better solutions, in terms of less transition time, to be developed. However, doing so would likely require a significant rework of the entire recompute process.

However, the most significant problem is that the worst performing model is the one representing the current IDEX physical construction. Therefore, without an alteration to the construction of IDEX printers, this algorithm is a non-ideal solution. However, changing the construction of IDEX printers is not far-fetched. Even with this algorithm as presented, a feasible printer could be constructed today with 20% improvement in print time. This is not revolutionary, especially given the added mechanical complexity, but represents an excellent starting point. If such a printer were to exist, the algorithms would certainly improve. Thus, while the results may not be encouraging on current printers, it is encouraging for the future of IDEX printing.

4.2 Tightening Constraints on the Solutions

There exist several interesting means by which to tighten the constraints of the various models. For example, the current prevailing model for IDEX printers has one hotend printing a support material, which is ultimately removed from the finished part, and the other hotend actually printing the part. There is no reason that this process cannot be parallelized. However, two minor issues arise in doing so. First, the data for segments to hotend must be encoded in the GCode file. This is trivial: since printers exist today with this behavior, there must be some encoding of this information out there presently. The second constraint arises in the algorithm itself, in preventing segments from being assigned to the wrong agent. It seems like this behavior should be relatively easy to integrate into the chain matching function. In doing so, it would enable the time performance improvements seen herein to be applied in a more traditional IDEX use case. In general, this would enable many interesting features, such as allowing two hotends to print concurrently with different colored materials on one part.

There are of course several other possible ways to alter the constraints. For example, if the outer surface of a part must be printed with one material by one hotend while the internals should use a cheaper material. Perhaps more interestingly is the same constraint but relaxed such that the first hotend can print the inside in order to minimize total print time. Of course, in this scenario the time the first hotend spends printing the internals should be minimized.

4.3 Expanding beyond Fused Deposition Modeling 3D printing

The solutions presented herein should be largely extensible. With GCode being a relatively common intermediary file for Computerized Numerical Control (CNC) machines, the applications of this algorithm are quite broad. Beyond even the 3D printing space, consider for a moment CNC Routers. These machines are largely the same as 3D printers, differing only in that they subtract material rather than add it. Thus, they too should be able to benefit from the IDEX/CODEX improvements shown in this research, or any future work. Since this code represents a GCode-to-GCode transformation, it should be a relatively trivial extension of these ideas.

4.4 Physical Constraint Informed Performance Improvements

Both the CODEX model and Current model are directionally biased. They are bound by physical constraints in the X and Y dimensions. Therefore, it is only natural to ask how the preparation of the GCode file influences the output. For example, could rotating the input GCode file improve the solution found? Could a heuristic be utilized to determine the optimal rotation efficiently? A challenge arises in that all Z-Layers would need to be transformed in the same manner; any heuristic would need to operate on the layers in aggregate. One such example could be determining axis of symmetry and aligning this axis with the Y-axis for the CODEX model or the X-axis for the Current model.

4.5 Improving the Recompute Time

There are two primary ways to improve the per-file recompute performance. The first solution involves multi-threading the recompute of a single file. Since each Z-Layer is recomputed independently, it should be possible to multi-thread each layer, thereby reducing the total wall time taken. Ultimately this was not helpful for this experiment as the CPU was already saturated by the quantity of files present in the dataset and the multi-processing of these files. However, in a more realistic scenario, in which a single file is to be processes, this could bring about significant performance improvement.

The second method of improving recompute time can be accomplished via hardware acceleration. Through profiling of the code, it was determined that roughly half of the processing time is spent deciding if two segments can be printed concurrently, a decision that is often repeated for the same pairs of segments. Therefore, if the printability of the segments were determined before chain matching and stored in a lookup table, the recompute time could be significantly reduced. Experimentally, this was verified by providing a collisions table computed a priori. However, the time to construct this table almost always exceeds the time saved by utilizing it. Thus, this new recompute process was untenable. One area of interest is then to improve the collision table generation. It is believed that this could be done via GPU based hardware acceleration. Initial experiments were performed using CUDA 11 on an NVIDIA RTX 3070 with promising results. However, the process suffered from numerous logical errors and could not be accurately implemented as of writing.

REFERENCES

[1] Ali, Md Hazrat, Shaheidula Batai and Dastan Sarbassov. "3D printing: A critical review of current development and future prospects." *Rapid Prototyping Journal* (2019).

[2] Baumann, Felix W. "Thirty Thousand 3D Models from Thingiverse." *MDPI Data* 18 January 2018. Web.

[3] BCN3D. *IDEX Technology*. 2021. Web. 3 April 2021. <https://www.bcn3d.com/technology/>.

[4] E3D-Online. "V6 Drawings." 24 June 2016. *E3D Online.* Document. 5 March 2021. <https://e3d-online.dozuki.com/c/V6_Drawings>.

[5] Hunter, J. D. "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering* 9.3 (2007): 90-95. web. <https://matplotlib.org/>.

[6] Microsoft. *Visual Studio Code*. n.d. Web. 19 March 2021. <https://code.visualstudio.com/>.

[7] mietek. *nc-gocde*. 10 October 2018. 19 March 2021. <https://marketplace.visualstudio.com/items?itemName=ML.nc-gcode>.

[8] Mwema, Fredrick Madaraka and Esther Titilayo Akinlabi. "Basics of Fused Deposition Modelling (FDM)." *Fused Deposition Modeling*. Springer, 2020. 1-15. Web.

[9] O'Donnell, Patrick A and Toms Lozano-Perez. "Deadlock-free and collision-free coordination of two robot manipulators." *Proceedings, 1989 International Conference on Robotics and Automation*. Scottsdale, AZ, USA: IEEE, 1989. 484-489. Web.

[10] Pereira, Guilherme, Fernando Gasi and Sérgio Ricardo Lourenço. "Review, Analysis, and Classification of 3D Printing Literature: Types of Research and Technology Benefits." *International Journal of Advanced Engineering Research and Science (IJAERS)* (2019): 167-187.

[11] Raise3D. *IDEX 3D Printer*. 2021. Web. 3 April 2021. <https://www.raise3d.com/e2/>.

[12] RepRap. *Main Page*. 24 September 2020. Web. 19 March 2021. <https://reprap.org>.

[13] Russell, Stuart and Peter Norvig. *Artifical Intelligence A Moder Approach*. Third Edition. Upper Saddle River: Pearson, 2010. Print.

[14] u/m47812. *Developing a new type of 3D printer, the CODEX (Collaborative Dual Extruder) printer*. 20 September 2020. Reddit. 5 March 2021. <https://www.reddit.com/r/3Dprinting/comments/iwdnin/developing_a_new_type_of_3d_printer_the_codex/>.

[15] Weiss, Mark Allen. *Data Structures and Algorithm Analysis in C++*. Fourth Edition. Upper Saddle River: Pearson, 2014. Print.