G-Code Based Toolpath Simulation for Predicting CNC Energy Consumption

By

Jonathan Anziani

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of
BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 2025

Authored by: Jonathan Anziani
Department of Mechanical Engineering
May 23, 2025

Certified by: A. John Hart
Class of 1922 Professor of Mechanical Engineering
Thesis supervisor

Accepted by: Daniel Frey
Professor of Mechanical Engineering
Undergraduate Officer

G-Code Based Toolpath Simulation for Predicting CNC Energy Consumption

by
Jonathan Anziani

Submitted to the Department of Mechanical Engineering

in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

ABSTRACT

Machining is an energy intensive process, and being able to model the energy consumption of machining would allow manufacturers to consider how to reduce their energy footprint. While many models have been developed for estimating energy consumption, they are not easily applicable or accessible to CNC machining, where the material removal rate is variable. This thesis develops a G-code based simulation that uses a voxel mesh to virtually recreate material removal, approximating the material removal rate at discretized points in the machining process. Using an energy consumption model and machine power data, material removal rates are related to the power consumption of machining the part. The simulation pipeline was validated using power data collected from literature, and for a constant material removal rate the model has shown average absolute error of 3.17% predicting power and 2.89% predicting specific energy consumption for simulated test geometries.

Thesis supervisor: A. John Hart
Title: Class of 1922 Professor of Mechanical Engineering

Acknowledgements

Thanks go out to all the Laboratory for Manufacturing and Productivity shop staff including Hannah Mishin, Joshua Ramos, and Wade Warman for their knowledge that became essential for completing this thesis. I would also like to thank Prof. John Hart for accepting to be my thesis supervisor and providing his time and expertise. Finally, I would like to thank Kaitlyn Gee for her dedication in steering me through this project.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Manufacturing involves large energy costs, utilizing much of the world's resources. The industrial sector, for example, uses 54% of the world's delivered energy, with only 7.4% of that energy originating from renewable sources [1]. Considering this large energy sink, being able to optimize product design and manufacture for minimum energy consumption can help reduce the high energy cost of manufacturing. Developing models that predict the energy cost of part production help allow product designers consider energy cost in their design process, and much research has been done modeling the energy cost of subtractive manufacturing.

Attempts have been made to analyze and predict the energy consumption of machining. These often come in the form of models that relate specific energy consumption to machining process parameters. Specific energy consumption (SEC) is the amount of energy consumed to remove a unit of volume of material, and the process parameter most often used is material removal rate (MRR). One of the earliest models relating SEC to MRR was posed by Gutowski et al. [2], noting the inverse relationship between SEC and MRR. Subsequent models built off of this concept, considering additional parameters in addition to SEC. For example, Li et al. [3] presented an improved SEC model that considers the impact of spindle speed on SEC, in addition to MRR.

While these energy models can be accurate, they are limited in several ways. Each machine tool will have different coefficients for the same model, so the data acquisition process would have to be repeated for every machine tool [3]. Workpiece material, tool material, tool wear, and coolant usage all impact the coefficients and accuracy of these types of models [4]. Additionally, only energy used to cut material is considered in these models, limiting their usability.

Looking more broadly at machining energy, some machine-tool models consider power contributions of the entire machine, which is valuable because a large amount of the energy expenditure is not associated with the machining of material. As an example, Kordonowy showed that 65.8% of a CNC's peak power consumption was variable on load and associated with machining while 33.4% was constant and associated with auxiliary components like tool changers, coolant pumps, and spindle efficiency [5]. Considering the aforementioned variability within machine tools, these power consumption shares vary significantly from machine to machine, so looking at the entire machine tool can provide an accurate assessment of the full energy cost of the machining process.

While the empirical SEC models have been shown to be accurate in predicting the energy consumption of simple constant MRR milling operations, CNC machining often consists of complex cutting geometries with variable MRR, so a more generalizable approach is necessary. A model that can predict the energy cost of a part based on the G-code of said part would be more applicable to CNC machining. Machine learning has been one approach to

predict the energy cost of an entire numerically controlled machine program, and these models can become very accurate. An artificial neural network model, for example, was developed that was achieved a maximum of 0.57% error for various slot, hole, and pocket cuts [6]. While this approach to energy consumption modeling may be very accurate, it requires large amounts of data. Recently a simulation approach has been developed, where the material removal process is virtually simulated and a power model is used to predict the energy consumption [7].

This thesis explores a simulation approach to predict the power profile and energy consumption of machining a part based on the G-code. The simulation is driven by an existing SEC model, generalizing these SEC models to be applicable in a continuous MRR use case.

Chapter 2: Methods

Model Design

At a high level, this simulation functions by using a voxel mesh to virtually recreate material removal and then using pre-existing SEC modeling research to estimate the energy cost at discretized points, allowing for a more representative modeling of the entire machining process when compared to the original models. The energy consumption simulation consists of toolpath generation, material removal simulation, material removal rate to specific energy consumption relation, and energy summation, as visualized in Figure 1. This simulation requires the G-Code used to machine the part, the feedstock geometry the part is machined from, empirically collected material removal rate and power data from the machine-tool system, the endmill radius, and the machine's rapid feedrate.

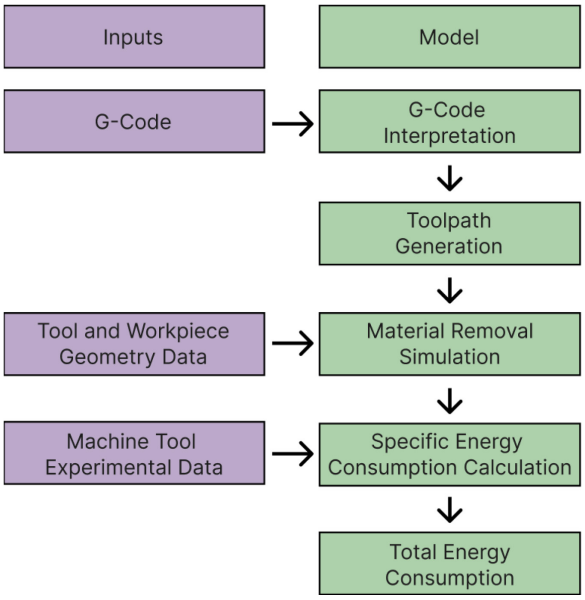


Figure 1: Simulation workflow overview

When the G-Code is inputted into the simulation, it is processed to remove any non-movement related text, such as comments and headers. Once this is done, the program reads each line and interprets the motion states, distance modes, start position, and end positions of each line. To simplify the model, the only the following commands were considered: G00/G01 (linear motion), G02/G03 (arc and helical motion), G90/G91 (distance modes), and G17/G18/G19 (plane modes). These model limitations simplified the development of the program without restricting the design possibilities of inputted parts.

To generate the toolpath, the model takes the start and end positions of each line and interpolates the path based on the motion state (rapid linear, linear, clockwise arc, and counterclockwise arc). Based on the movement type, the program calculates the distance the endmill travels to reach the end position, and using the feedrate, the runtime of the line dt_n is calculated. The number of points used to interpolate the path is a user defined value

Linear Motion

For a linear motion operation (G00 and G01), the program calculates the length of the line between the start and end positions evaluating the Euclidian 2-norm of the difference between the two points. Given the feedrate of the operation from the G-Code interpretation (or the rapid feedrate) and the distance travelled in the operation, the program calculates the runtime of that line of G-Code dt_n. A linearly-spaced array of the runtime is generated with a size equal to the number of interpolation steps and are interpolated using linear interpolation to approximate the toolpath.

Arc and Helical Motion

For arc and helix operations (G02 and G03), the program first finds determines the center of the arc turn. For an offset-based arc operation, this center point is the offsets added onto the start point. For a radius-based turning operation, the center point is decided by finding the circular intersection of two circles using the radius, r, defined in the line of G-code, with their centers being the start and endpoints of that line of G-Code. The point of intersection of the two circles is the center of the milling radius, (C_X, C_Y).

Using the center, the program finds the vector from the center to the start and the vector from the center to the end. It then calculates the positive counterclockwise angle from horizontal for each of the two vectors using the four-quadrant inverse tangent. By subtracting the center to end angle by the center to start angle, θ,the counterclockwise change in angle, $\Delta\theta$, over the operation is found. For a G03 operation, this is accurate, but for a G02 clockwise operation, the is subtracted from 2π.

Runtime for the operation is calculated. To do this, the height of the helix, h, is found as the absolute change in axial distance from the start to end point, which is zero for a planar arc,

this value is just zero. Then, the number of turns the arc travels over the operation, n_{turns}, is taken from the G-Code defined by the letter "P" and is zero if undefined. The pitch, *p*, of the helix is calculated using the following equation:

$$p = \frac{h}{2\pi(n_{turns} - 1) + \Delta\theta}$$

The total distance travelled for an arc or helix is then calculated using the following equation:

$$length = n_{turns}\sqrt{(2\pi r)^2 + p^2}$$

This total length gets divided by the feedrate of the operation to get the runtime dtn and a linearly-spaced array of the runtime, T, is generated with a size of P. The arc toolpath is generated by interpolating T over using the following equation for a counterclockwise path:

$$x = r \times \cos(\theta + \frac{2\pi(n_{turns} - 1) + \Delta\theta}{dt_n} \times T) + C_x;$$

$$y = r \times \sin(\theta + \frac{2\pi(n_{turns} - 1) + \Delta\theta}{dt_n} \times T + C_y;$$

For clockwise paths, the equations used are the following:

$$x = r \times \cos(\theta - \frac{2\pi(n_{turns} - 1) + \Delta\theta}{dt_n} \times T) + C_x;$$

$$y = r \times \sin(\theta - \frac{2\pi(n_{turns} - 1) + \Delta\theta}{dt_n} \times T + C_y;$$

The interpolated points for all operations are connected sequentially into in one toolpath with their respective timesteps. Figure 2 shows a sample toolpath interpolated by the simulation.

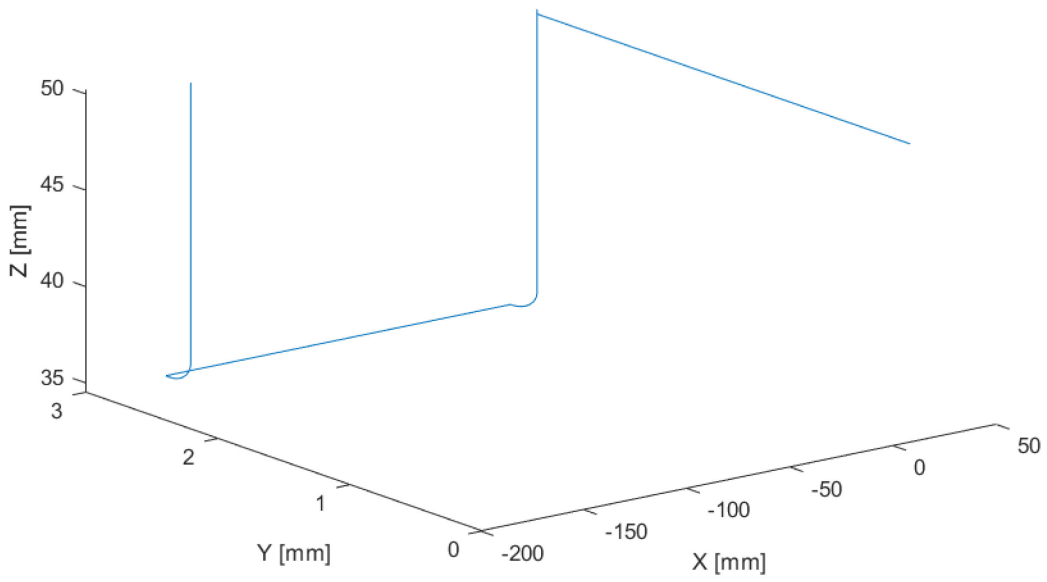


Figure 2: A toolpath interpolated by the simulation

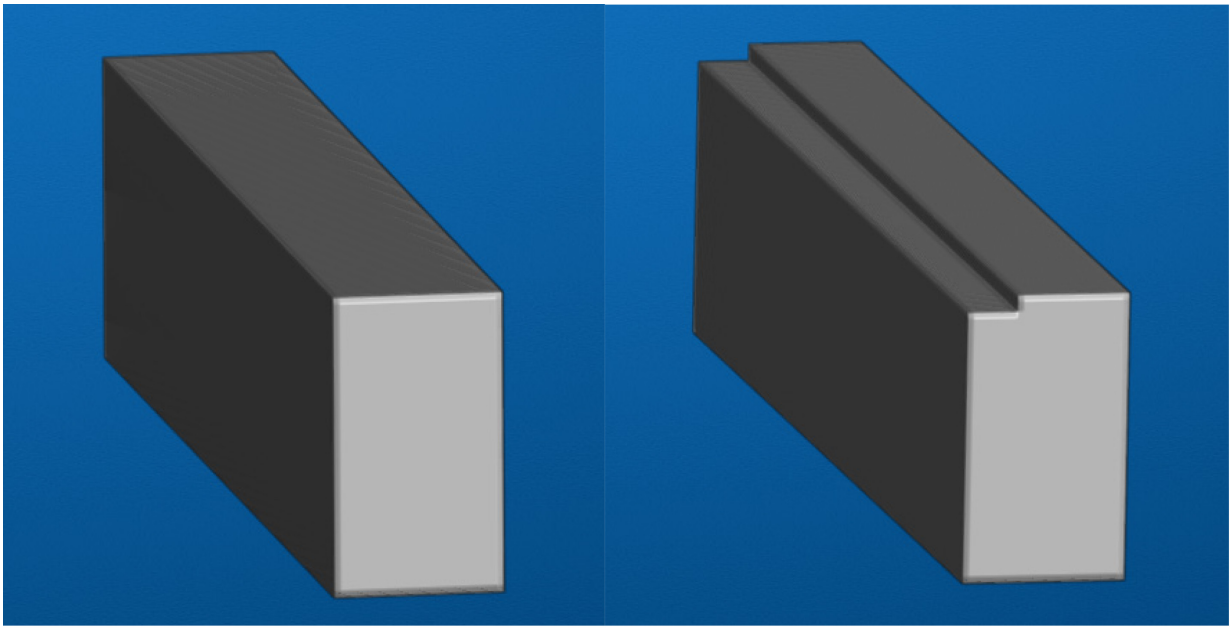
Material Removal Simulation

Material removal is simulated by first uploading a 3D STL model of the stock material that will be milled by the CNC to produce the final part generated from Fusion 360. Using a raytracing algorithm library, the stock material 3D file is converted into a voxel mesh with a user-inputted resolution [8].

The simulation then runs through the each of the points in the interpolated toolpath sequentially with a virtual endmill of with a user-inputted radius. If the virtual endmill comes in contact with the majority of a voxel as determined by logical expressions used on the mesh, the voxel is removed from the grid, replicating material removal. Each voxel represents a volume equal to the cube of the grid resolution, so each voxel removed is equivalent to removing that amount of volume, so a .1 mm resolution would remove .001 mm^3 of material.

Moving from one point to another, the material removal simulation makes use of five geometries to approximate the milling operation. The first two are cylinders, with their bottom faces centered on the start and end positions of that specific timestep. The next two geometries include an oblique cylinder connecting the bottom faces of the previous cylinders and a second oblique cylinder connecting the top faces. The final geometry is an oblique rectangular prism running parallel to the direction of motion and connecting the rectangular cross-sections of the two cylinders. If the majority of a voxel is found within these geometries during a timestep based on logical expressions defining the bounds of these geometries, it is

removed from the mesh. This process is repeated for every point along the toolpath, saving the change in volume in an array which can be converted into MRR by dividing by the timestep. Figure 4 shows a voxel mesh before and after a material removal simulation.



a) b)

Figure 3: A sample of the voxel mesh a) before and b) after material removal

Specific Energy and Power Calculation

A MRR and spindle-speed based model is used to relate MRR and spindle speed to SEC and power. The model this simulation uses are proposed in Zhou et al. [9] and are shown in the equations below. Although this specific model was chosen, the simulation can be replaced by any MRR and spindle speed-based model, allowing this pipeline to be easily scalable for future model improvements.

$$Power = \ c_1 + c_2 \times n + c_3 \times n^{c_4} \times MRR \text{ [9]}$$

$$SEC = \ c_1' \times n^{c_2'} + \frac{c_3' \times n}{MRR} + \frac{c_4'}{MRR} \text{ [9]}$$

To collect the coefficients for the model, power data taken directly from the machine is required. More specifically, a table power data extracted from milling operations at varying MRR and spindle speeds can be used to curve fit the data to power model. Using the SEC, cutting power, and MRR relation below, the SEC values for the measured milling operations can be calculated.

$$SEC = \frac{P_{cut}}{MRR}$$

With the SEC, MRR, and spindle speed data, the coefficients for SEC model can be found. Each curve fit is done using nonlinear least squares, and the new curve fits would have to be done for every machine, tool, and material combination that is meant to be simulated. With the curve fitted equations the model finds the SEC and power at every timestep using the MRR and spindle speed arrays.

Individual cutting energy contributions are then be through element-wise multiplication of the SEC and volume change arrays. These individual contributions can be summed up to approximate the total cutting energy consumption.

Additional energy consumptions can also be found with this simulation pipeline. Using the power curve-fit equation, the power draw of air-cutting can be approximated using the spindle speed. Multiplying the power draw by the timesteps provides the individual contributions of air-cutting.

The simulation also flags when the coolant is turned on or off using M08 and M09. After finding the power draw of the coolant pump, the energy contribution of using the coolant pump can be found by multiplying the power draw with the runtime. Similar flags can be set to additional auxiliary functions to track their energy contributions as well, developing a prediction model of the entire machining process.

Validation

Experimental validation was not possible due to complications in installing the power measuring instrument, so validation made use of existing experimental data from Zhou et al. (2017) [9]. A proposed experimental validation plan is presented in Appendix A.

Using nonlinear least squares on the material removal rate, spindle speed, and cutting power values from Table 5 of Zhou et al. [9], the coefficients used in for the power model validation were found. SEC is then calculated from the power material removal rate. Following this, the coefficients for SEC model were found by nonlinear least squares with SEC values derived from the prior power and MRR values.

The nine milling operations described in Table 7 in Zhou et al. [9] were recreated in Fusion 360 using the same depth of cut, width of cut, length of cut, feedrate, spindle speed, and endmill diameter as described in the paper. These milling operations were then exported as G-code, and the stock material was exported as a 3D STL file.

The G-Code and STL are then run through the material removal simulation, with the curve-fit equations being used to predict the SEC based on the MRR. These predicted SEC values are then compared to the measured SEC vales from Zhou et al. [9]

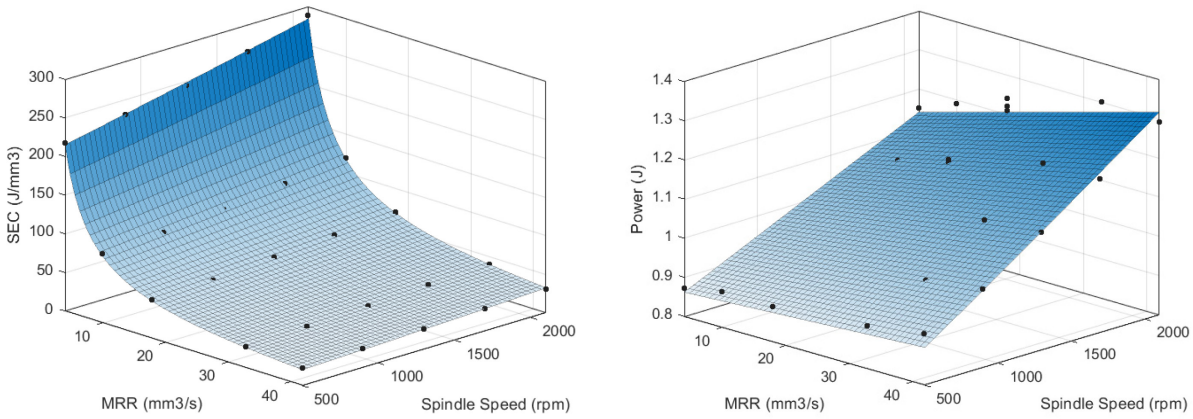
Chapter 3: Results and Discussion

Model Validation

The results of the curve-fits for SEC and power models can be found in Table 1 and visualized in Figure 4.

Table 1: Curve fitting results and goodness of fit

	c1	c2	c3	c4	SSE	R^2	DFE	adj-R^2	RSME
Power	0.78	0.00017	8.06E-07	1.13	0.012	0.98	31	0.98	0.019
SEC	5.39E-06	1.82	0.15	779	71.64	1.00	31	1.00	1.52



a) b)

Figure 4: Curve fit visualizations for a) SEC model and b) power model

Figure 5 shows a sample time-series simulation of a milling operation with MRR of 18 mm^3/s. The average MRR over the entire duration of the sample is 16.67 mm^3/s (-7.37% compared to the intended 18%), but when not considering the beginning and end points the average becomes 18.01 mm^3/s (0.06% error). There is a variable MRR as the endmill enters and exits the workpiece due to it cutting less material at the same feed rate, so the additional error is a result of the simulation modeling this changing MRR. These results indicate that a numerical computational pipeline that discretizes the G-code toolpath, grounded in the SEC framework of Zhou et al. [9], has the potential to accurately estimate energy consumption of milling.

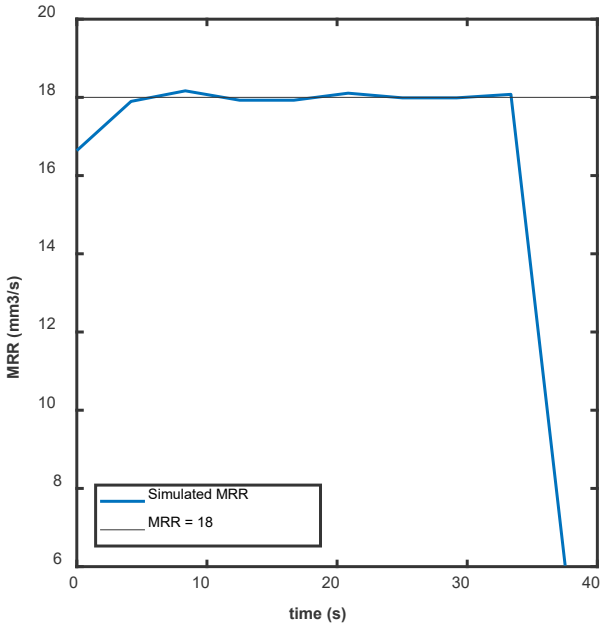


Figure 5: Sample MRR simulation

Table 2 compares the expected and simulated MRR, SEC, and power using the nine milling operations described in Table 7 in Zhou et al. [9] and while ignoring the endmill's entry and exit from the material. With an average absolute MRR error of 0.05% and a maximum error of 0.11%, the simulation is capable of accurately replicating a constant MRR process. The MRR error was additionally shown to be correlated to the depth of cut, with 3 mm, 6 mm, and 9 mm depth of cuts having an error of 0.00%, -0.11%, and 0.06% respectively. The average absolute SEC error of 2.89% and the average absolute power error of 3.17% indicates that the simulation can reasonably predict the SEC and power for a timestep.

Table 2: MRR, power, and SEC error using 10 points of interpolation and 0.5 mm resolution with expected values from Zhou et al. [9]

Test No.	Spindle Speed (rpm)	Depth of Cut (mm)	Width of Cut (mm)	Feed Rate (mm/min)	Expected MRR (mm^3/s)	Model MRR (mm^3/s)	MRR Error (%)	Expected Power (kW)	Model Power (kW)	Power Error (%)	Expected SEC (J/mm^3)	Model SEC (J/mm^3)	SEC Error (%)
1	500	3.0	0.50	120	3.0	3.00	0.00	0.83	0.86	3.73	276.67	286.61	3.59
2	1300	6.0	1.50	120	18.0	17.98	-0.11	1.03	1.04	0.86	57.22	57.31	0.16
3	2100	9.0	2.00	120	36.0	36.02	0.06	1.36	1.29	-5.18	37.78	36.85	-2.45
4	2100	6.0	0.50	180	9.0	8.99	-0.11	1.17	1.16	-0.48	130.00	136.12	4.71
5	500	9.0	1.50	180	40.5	40.52	0.06	0.95	0.90	-5.77	23.46	21.62	-7.81
6	1300	3.0	2.00	180	18.0	18.00	0.00	1.03	1.04	0.70	57.22	57.25	0.06
7	1300	9.0	0.50	240	18.0	18.01	0.06	1.01	1.04	2.87	56.11	57.22	1.99
8	2100	3.0	1.50	240	18.0	18.00	0.00	1.24	1.21	-2.74	68.89	67.79	-1.60
9	500	6.0	2.00	240	48.0	47.95	-0.11	0.96	0.90	-6.22	20.00	19.26	-3.69

The simulation results were then compared to the accuracy of other models using values from Zhou et al. [9] and Li et al. [3] Table 3 compares the simulation to models proposed by Zhao et al. and Li et al. Comparing the power, the simulation is more accurate than Li's model but equal to that of Zhou, which is expected as the simulation is built off of the Zhou's model. Comparing the SEC however, the simulation is less accurate than Zhou's model. Potential causes of the disparity between Zhou's model and the simulation can be the use of different curve fitting coefficients and the impacts of the MRR error, but more data is required to find the cause. Nonetheless, the simulation has comparable accuracy to existing models.

Table 3: Power and SEC simulation accuracy comparison to other models

Test No.	Expected Power (kJ)	Expected SEC (J/mm^3)	Model	Predicted power (kJ)	Error (%)	Predicted SEC (J/mm^3)	Error (%)
3	1.36	37.78	Li et al. [3]	1.26	-7.35	33.47	-11.41
			Zhou et al. [9]	1.29	-5.15	37.45	-0.87
			Simulation	1.28	-5.88	36.85	-2.46
4	1.17	130	Li et al. [3]	1.18	0.85	127.64	-1.82
			Zhou et al. [9]	1.16	-0.85	130.1	0.08
			Simulation	1.16	-0.85	136.12	4.71

Sensitivity Analysis

To see how the estimations change when adjusting the model parameters, the simulation was run while varying the number of interpolation points and the simulation resolution using the cutting parameters from test 7 in Table 2. Figure 6 shows the errors in MRR, SEC, and power as the number of interpolation points changes. As the number of interpolation steps increases, the MRR error decreases. Power error remains relatively constant at an average of 2.86%. SEC, however, increases in error with interpolation steps and has a clear inverse correlation with the MRR error.

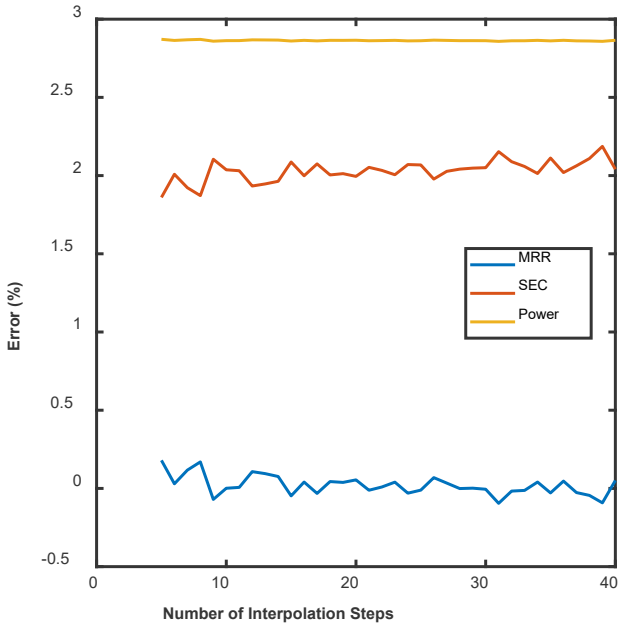


Figure 6: MRR, SEC, and power error results at varying number of interpolation steps

To further investigate the relationship between the interpolation and MRR, Figure 7 shows how the simulated MRR changes between 10 and 40 interpolation steps. Both simulations have the simulated MRR values oscillate around the actual MRR value, but for higher MRR values, the amplitudes of these oscillations become larger. To demonstrate this, Figure 8 compares the MRR standard deviation of the stabilized portion of the milling operation, starting at 0.02 mm^3/s for 5 steps and reaching a local maximum of 0.46 mm^3/s at 39 steps.

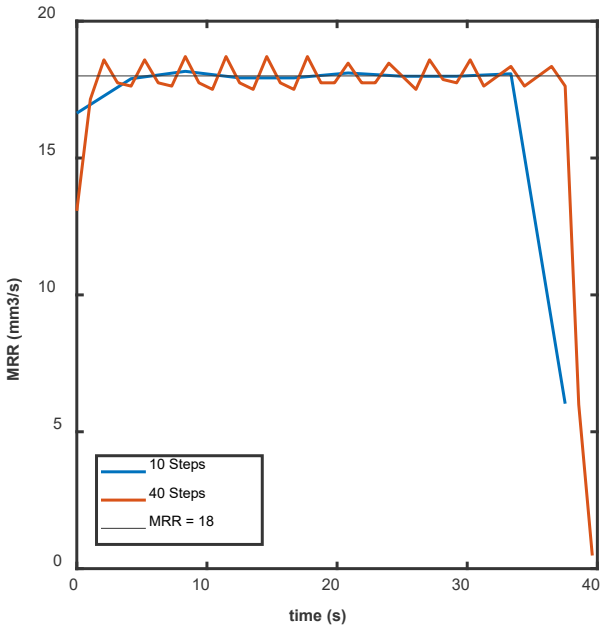


Figure 7: Comparison between MRR simulation for 10 and 40 interpolation steps

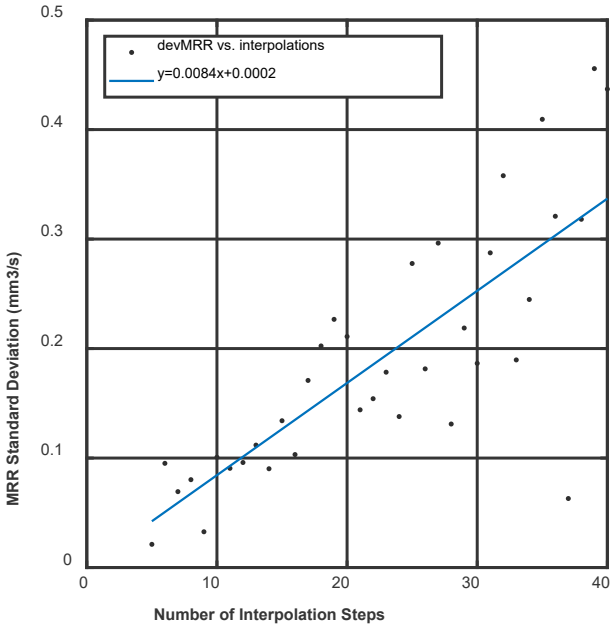


Figure *8*: Simulated MRR standard deviation at varying number of interpolation steps

Figure 9 shows the error in MRR, SEC, and power when manipulating the model resolution variable, and all variables are shown to be sensitive to changes in resolution. The errors are the lowest at the resolutions 0.1 mm, 0.125 mm, 0.25 mm, and 0.5 mm. This is because at these resolutions, the simulated endmill operation aligns precisely within the voxel grid, allowing the simulation to accurately assess if the endmill is contacting material. Further

testing can analyze how the endmill's alignment with the grid impacts the MRR, SEC, and power accuracy of non-linear milling, where the tool's alignment is constantly changing.

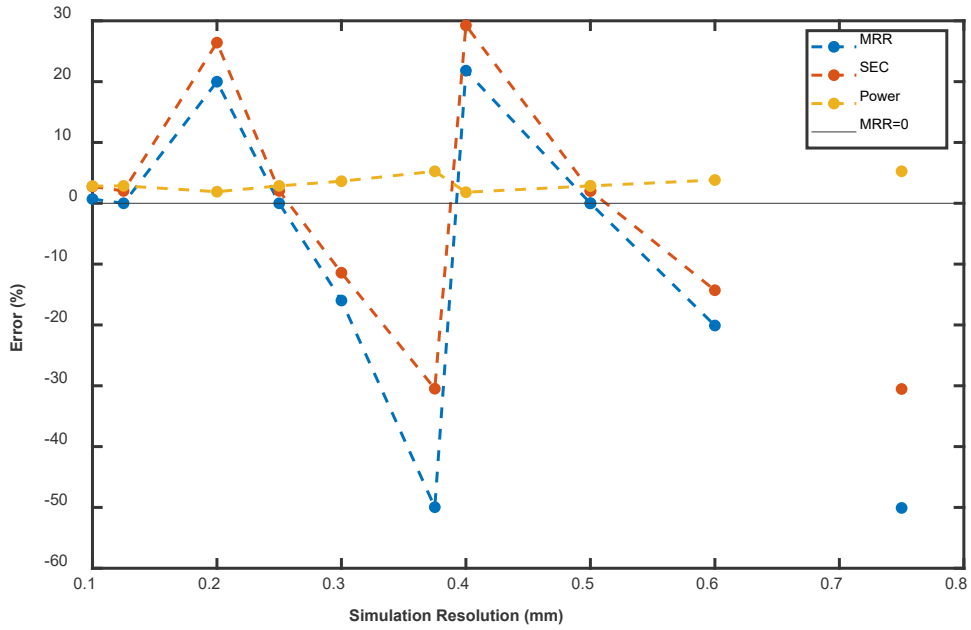


Figure 9: MRR, SEC, and Power error at varying simulation resolutions

To investigate if the modeled entry and exit into a milling operation significantly alters its energy prediction, the simulation's cutting energy prediction was compared to a prediction considering only a constant MRR. The simulation's energy prediction was calculated by multiplying the SEC for each timestep by the volume of material removed at that timestep and taking the sum of all the individual energy contributions. The constant MRR energy prediction was calculated by multiplying the volume of material removed over a milling operation by the SEC measured in Zhou et al. [9] Table 4 shows the results of this comparison. On average, the simulation differs from the constant MRR prediction by 4.16%. Determining if the simulation is an accurate energy predictor relative to reality would require dedicated experiments.

Table 4: Simulated energy prediction compared to constant MRR energy prediction

Test No	Volume Cut (mm3)	Expected SEC (J/mm3)	Constant MRR Energy (J)	Model energy (J)	Difference (J)	Absolute Difference (%)
1	232.5	276.67	64325.08	68782.55	4457.48	6.93
2	1395	57.22	79824.69	82763.96	2939.27	3.68
3	2790	37.78	105400.6	109368.8	3968.22	3.76
4	465	130.00	60450	62454.78	2004.78	3.32
5	2092.5	23.46	49083.77	48640.78	-442.99	0.90
6	930	57.22	53216.46	54881.22	1664.76	3.13
7	697.5	56.11	39137.42	42832.57	3695.15	9.44
8	697.5	68.89	48050.08	48659.97	609.89	1.27
9	1860	20.00	37200	35343.99	-1856.01	4.99

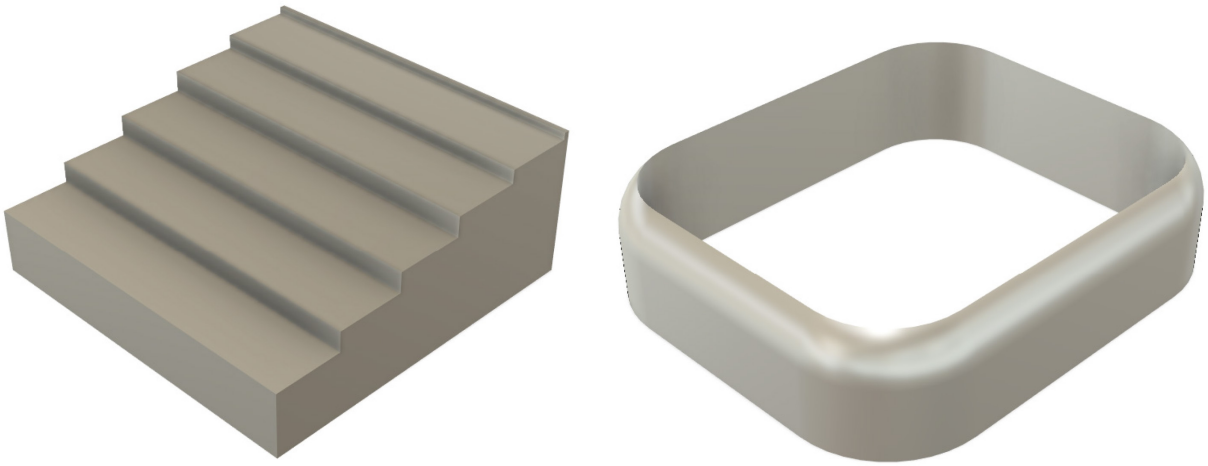
Chapter 4: Conclusion

This thesis develops a numerical pipeline for estimating energy consumption based on machine, material, and tool-specific inputs. By discretizing the G-Code toolpath and simulating MRR, this pipeline is able estimate the SEC with an average of 2.89% error relative to empirical data. These results suggest that numerical estimation of energy consumption is a viable approach to expedite life cycle assessments for machining operations, although future work is required to extend to other materials, machines, and cutting operations and to evaluate its performance on more complex geometries. Future planned experiments to test these properties of the model are proposed in Appendix A. Ultimately the model contributes to a larger body of numerical and computational tools aimed at empowering the next generation of product designers and manufacturing engineers to design holistically for sustainability.

Appendix A: Proposed Method for Further Validation

Additional work can be done to validate the accuracy of the simulation using more complicated geometry with complex cutting operations (i.e. variable MRR) and assessing the accuracy of the energy prediction. Additionally, investigating the simulation's performance on different materials would demonstrate the potential applicability with high SEC, industrial-grade materials like titanium. To do this, additional data must be empirically collected, and this section proposes an experimental plan to do so.

The validation will be done by machining two geometries, both in aluminum 6061 and Grade 5 titanium. The first geometry is a staircase-like shape used to curve-fit the SEC and power consumption at various MRRs and spindle speeds. The second geometry is inspired by an Apple Watch case and will be used to compare the power consumption of milling to the predicted output of the model. These proposed geometries are shown in Figure 9.



a) b)

Figure 10*: Proposed geometries for a) curve fitting and b) comparison and validation*

Depth of cut and width of cut values for the staircase part were determined to get a large range of roughing and finishing parameters for curve-fitting. Spindle speed and feed rates were determined by inputting the material, tool, depth of cut, and width of cut parameters into a machining assistance tool FSWizard [10]. For aluminum, the feed rates were multiplied by 0.7 to collect more samples per milling operation. Additionally, spindle speeds for aluminum were varied between 7000, 7250, and 7500 to provide a range of values for spindle speed. Titanium values remained unchanged from the FSWizard suggestions. The machining parameters were

found considering an uncoated 3-flute 6mm diameter carbide endmill for aluminum and an AlCrN coated 4-flute 6mm diameter carbide endmill for titanium.

The staircase parts used for constant calibration are made to curve fit datapoints to the below equation. Each of the five "steps" in the staircase consists of three cuts at three width of cuts (3mm, 2mm, and 1mm) leading to 15 different depth of cut and width of cut combinations. These 15 machining parameter combinations are shown in Table 5 for aluminum and Table 6 for titanium.

The "Apple Watch" part design was chosen due to having several locations with variable MRR while machining. The large pocket and the fillets allow for continuous changing of MRR Additionally, there is also a wide range of MRRs tested with this design, with high MRR being tested during the roughing passes and low MRR being tested during the finishing passes. Also, by using a design influenced by an actual manufactured project, it can directly show relevance to industrial manufacturing.

The power can be measured with the Victron VM-3P75CT power meter which was installed in the MIT LMP shop. This power meter was chosen due to its sampling rate, resolution, and power rating. The 10 samples per second sampling rate can show fast fluctuations in power across the varying MRR. The ±0.001 kW resolution allows the power to be measured at fine intervals, making the data more useful for curve-fitting and model validation. Its power rating of 265V and 75A of three-phase power allows this power meter to be compatible with the HAAS VF-OE Vertical Machining Center.

Table *5: Proposed aluminum experimental cutting parameters*

Depth of Cut (mm)	**Width of Cut (mm)**	**Feed (mm/min)**	**MRR (cm^3/min)**	**Spindle (rpm)**
12	3	345.07	12.42	7500
12	2	531.67	12.76	7250
12	1	792.79	9.51	7000
9	3	460.1	12.42	7500
9	2	708.89	12.76	7250
9	1	792.79	7.14	7000
6	3	690.15	12.42	7500
6	2	813.28	9.76	7250
6	1	792.79	4.76	7000
3	3	800.88	7.21	7500
3	2	813.28	4.88	7250
3	1	792.79	2.38	7000
1	3	800.88	2.4	7500
1	2	813.28	1.63	7250
1	1	792.79	0.79	7000

Table *6: Proposed titanium experimental cutting parameters*

Depth of Cut (mm)	Width of Cut (mm)	Feed (mm/min)	MRR (cm^3/min)	Spindle (rpm)
12	3	87.13	3.14	2317
12	2	96.15	2.31	2434
12	1	141.36	1.7	2458
9	3	87.13	2.35	2317
9	2	96.15	1.73	2434
9	1	188.48	1.7	2458
6	3	87.13	1.57	2317
6	2	138.37	1.66	2434
6	1	232.05	1.39	2458
3	3	165.28	1.49	2317
3	2	192.31	1.15	2434
3	1	231.05	0.7	2458
1	3	174.26	0.52	2317
1	2	192.31	0.38	2434
1	1	232.05	0.23	2458

Appendix B: Simulation Code

```
clear all
```

Read Files

```
%Get Gcode and turn it a table
gcode = uigetfile('*.txt;*.nc,','Select the GCODE file');
steps  = readlines(gcode); %table of gcode
Nmax=height(steps);
steps = regexprep(steps,'\([^)]+\)',''); %Remove comments

%read stock STL
stl = uigetfile('*.stl','Select the stock stl file');
[stlcoords,~,~] = READ_stl(stl);

%read SEC vs Spindle speed vs MRR table
DATA = readtable("table.png.csv");
MRR_mm3_s=table2array(DATA(:,"MRR_mm3_s_"));
Power_kW=table2array(DATA(:,"Pcut_kW_"));
n_rpm=table2array(DATA(:,"n"));

%create a fit line for the specific energy of cut
[fitresultSEC,fitresultPower,~]=createFit(MRR_mm3_s,n_rpm,Power_kW);
coefficientsSEC=coeffvalues(fitresultSEC);
coefficientsPower=coeffvalues(fitresultPower);

%Set resolution(s) and number of interpolation points
resolutions=.5;
interpolations=10;

home=[8,0,50]; %location of home in CNC
radii = 6; %radius of endmill
heights = 100; %height of endmill

%start data loop
for q=1:length(interpolations)

clearvars -except interpolations spiderMRR spiderPower spiderSEC q gcode steps Nmax stl stlcoords
coefficientsSEC coefficientsPower resolutions varSEC varPower varMRR
```

Initialize Constants

```
global RapidFeed
RapidFeed=1000; %rapid feed in CNC
global radius
global nTurns
global angle
global angle1
global Center


% Set grid resolution (smaller values give higher accuracy)
grid_resolution_mm = resolutions;
```

Make a table of Gcode

```
%make empty Gcode Table
alphabet = num2cell('A':'Z');
```

```
 varTypes= cell(1, 26);
 varTypes(:) = {'double'};
 varTypes(7) = {'string'}; % make G a string because multiple Gs
 GcodeTable = table('Size',[Nmax,26],'VariableTypes',varTypes,'VariableNames',alphabet);

 %fill with dummy string  to fix string error
 GcodeTable(:,"G") = [num2cell(repmat("A",1,height(GcodeTable(:,"G"))))]';

 %make sorted Gcode table
 for i=1:Nmax
     currentLine = steps{i,:}; %get current Gcode step
     split=regexp(currentLine,'\s', 'split'); %split codes in line
     for j=1 : length(split)
         splitLine(j)=split(j);
     end
     numberChanges=length(splitLine);
     letterString = 'G'; %make a string of every letter in code (ignoring G)

     %fill in codes per letter
     for j = 1 : numberChanges
         if ~isempty(splitLine{j}) %account for empty
             currentLetter = splitLine{j}(1);
             if currentLetter ~= "G"
                 GcodeTable{i,currentLetter}=str2double(splitLine{j}(2:end)); %fill in number after non G
letter
                 letterString = append(letterString,currentLetter); % add letter to string
             else
                 GcodeTable{i,currentLetter}=GcodeTable{i,currentLetter}+','+splitLine{j}(2:end); %fill
Gstring with G
             end
         end
     end
     clear splitLine currentLetter
     notLetterString = alphabet(~ismember(alphabet, num2cell(letterString))); %make string of unused
words
     for k=1 : length(notLetterString)
         GcodeTable{i,notLetterString(k)}=nan; %keep track of unused words with NaN
     end
 end
```

Simulate toolpath to produce coordinates, path length, run time

```
 %run through G code interpretation
 global Gmatrix %global matrix of G state [MoveType,DistanceType]
 Gmatrix=[0,91,17];
 for N=1:Nmax
     [Mov,Dist,Pla] = interpretGs(N,GcodeTable);
     Gmatrix(N,:)= [Mov,Dist,Pla];
 end

 %create table of start, end, length, and end time for each line
 coordNames = ["X1", "Y1", "Z1", "X2", "Y2", "Z2", "Length", "End Time"];
 coordTypes = ["double", "double", "double", "double", "double", "double", "double", "double"];
 coordinates_mm = table('Size', [Nmax,length(coordTypes)],'VariableNames', coordNames, 'VariableTypes',
coordTypes);
 coordinates_mm{:,:} = nan;
 coordinates_mm{1,"X1"} = home(1);
 coordinates_mm{1,"Y1"} = home(2);
 coordinates_mm{1,"Z1"} = home(3);

 %go through each line and build toolpath
 global CurrentMove
 CoordinatesOverTime_mm = table(home(1),home(2),home(3),0,VariableNames=["x","y","z",'dt']);
 spindleSpeed_rpm=[];
 coolant_N=[0]; %logical array of if coolant is on
 CoolantOverTime=[0];
 for N=1:Nmax %for each line update the coordinate values based on gcode path
```

```
     [NewCoordinates_mm,len,time_s,spindle_rpm] = DeterminePath(N,GcodeTable,coordinates_mm); %update
final position of current line
     coordinates_mm{N,"X2"}=NewCoordinates_mm(1);
     coordinates_mm{N,"Y2"}=NewCoordinates_mm(2);
     coordinates_mm{N,"Z2"}=NewCoordinates_mm(3);

     coordinates_mm{N,"Length"}=len; %update length and start positions
     coordinates_mm{N+1,"X1"} = coordinates_mm{N,"X2"};
     coordinates_mm{N+1,"Y1"} = coordinates_mm{N,"Y2"};
     coordinates_mm{N+1,"Z1"} = coordinates_mm{N,"Z2"};

     %update end time
     if N==1
         coordinates_mm{N,"End Time"}=time_s;
     else
          coordinates_mm{N,"End Time"}=time_s+coordinates_mm{N-1,"End Time"};
          coolant_N(N)=coolant_N(N-1);
     end

     if GcodeTable{N,"M"}==8
         coolant_N(N)=1;
     elseif GcodeTable{N,"M"}==9
         coolant_N(N)=0;
     end

     if isempty(CurrentMove)

     elseif CurrentMove=="Linear"

          if Gmatrix(N,1)==0 %if rapid
          %define timestep for operation
          numPoints=1; %number of points per operation
          dt=(time_s/numPoints)*ones(numPoints+1,1);
          t_s= 0:dt(1):time_s; %make an array from start to end time

          else
          %define timestep for operation
          numPoints=interpolations(q); %number of points per operation
          dt=(time_s/numPoints)*ones(numPoints+1,1);
          t_s= 0:dt(1):time_s; %make an array from start to end time
          end

     %develop timestep-dependent toolpath array
     if time_s~=0
         x = coordinates_mm{N,"X1"}+(coordinates_mm{N,"X2"}-coordinates_mm{N,"X1"})/time_s*t_s;
         y = coordinates_mm{N,"Y1"}+(coordinates_mm{N,"Y2"}-coordinates_mm{N,"Y1"})/time_s*t_s;
         z = coordinates_mm{N,"Z1"}+(coordinates_mm{N,"Z2"}-coordinates_mm{N,"Z1"})/time_s*t_s;
         x=x';
         y=y';
         z=z';

         %add toolpath to list of coordinates
         if N==1
             CoordinatesOverTime_mm = table(x,y,z,dt);
             CoolantOverTime = coolant_N(N)*ones(size(x));
         else
             CoordinatesOverTime_mm = vertcat(CoordinatesOverTime_mm,table(x,y,z,dt));
             CoolantOverTime = vertcat(CoolantOverTime,coolant_N(N)*ones(size(x)));


         end
     else
     end

     elseif CurrentMove=="CCW Arc"
         %define timestep for operation
         numPoints=ceil(2*len/radii); %number of points per operation
         dt=(time_s/numPoints)*ones(numPoints+1,1);
          t_s= 0:dt(1):time_s; %make an array from start to end time
```

```
        %develop timestep-dependent toolpath array
        if Gmatrix(N,3)==17
        x = radius*cos(angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(1);
        y = radius*sin(angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(2);
        z=coordinates_mm{N,"Z1"}+(coordinates_mm{N,"Z2"}-coordinates_mm{N,"Z1"})/time_s*t_s;
        elseif Gmatrix(N,3)==18
        x = radius*cos(angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(1);
        z = radius*sin(angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(2);
        y=coordinates_mm{N,"Y1"}+(coordinates_mm{N,"Y2"}-coordinates_mm{N,"Y1"})/time_s*t_s;
        elseif Gmatrix(N,3)==19
        y = radius*cos(angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(1);
        z = radius*sin(angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s)+Center(2);
        x=coordinates_mm{N,"X1"}+(coordinates_mm{N,"X2"}-coordinates_mm{N,"X1"})/time_s*t_s;
        end
        x=x';
        y=y';
        z=z';

        %add toolpath to list of coordinates
        if N==1
            CoordinatesOverTime_mm = table(x,y,z,dt);
            CoolantOverTime = coolant_N(N)*ones(size(x));
        else
            CoordinatesOverTime_mm = vertcat(CoordinatesOverTime_mm,table(x,y,z,dt));
            CoolantOverTime = vertcat(CoolantOverTime,coolant_N(N)*ones(size(x)));
        end

    elseif CurrentMove=="CW Arc"
        %define timestep for operation
         numPoints=ceil(2*len/radii); %number of points per operation
         dt=(time_s/numPoints)*ones(numPoints+1,1);
         t_s= 0:dt(1):time_s; %make an array from start to end time

        %develop timestep-dependent toolpath array
        if Gmatrix(N,3)==17
        x = radius*cos((angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(1);
        y = radius*sin((angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(2);
        z=coordinates_mm{N,"Z1"}+(coordinates_mm{N,"Z2"}-coordinates_mm{N,"Z1"})/time_s*t_s; %check if
works

        elseif Gmatrix(N,3)==18
        x = radius*cos((angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(1);
        z = radius*sin((angle1+(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(2);
        y=coordinates_mm{N,"Y1"}+(coordinates_mm{N,"Y2"}-coordinates_mm{N,"Y1"})/time_s*t_s; %check if
works

        elseif Gmatrix(N,3)==19
        y = radius*cos((angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(1);
        z = radius*sin((angle1-(2*pi*(nTurns-1)+angle)/time_s*t_s))+Center(2);
        x=coordinates_mm{N,"X1"}+(coordinates_mm{N,"X2"}-coordinates_mm{N,"X1"})/time_s*t_s; %check if
works
        end
        x=x';
        y=y';
        z=z';

        %add toolpath to list of coordinates
        if N==1
            CoordinatesOverTime_mm = table(x,y,z,dt);
            CoolantOverTime = coolant_N(N)*ones(size(x));
        else
            CoordinatesOverTime_mm = vertcat(CoordinatesOverTime_mm,table(x,y,z,dt));
            CoolantOverTime = vertcat(CoolantOverTime,coolant_N(N)*ones(size(x)));
        end
    end
    if isempty(spindle_rpm)
        spindle_rpm=0;
    end
```

```
    spindleSpeed_rpm=[spindleSpeed_rpm;spindle_rpm.*(ones(height(CoordinatesOverTime_mm)-
height(spindleSpeed_rpm),1))];
 end
 CoordinatesOverTime_mm=table2array(CoordinatesOverTime_mm); %turn into array

 %plot toolpath
 figure
 plot3(CoordinatesOverTime_mm(:,1),CoordinatesOverTime_mm(:,2),CoordinatesOverTime_mm(:,3))
 title("Toolpath")
 xlabel("X [mm]")
 ylabel("Y [mm]")
 zlabel("Z [mm]")

 %generic stuff for testing
 N=length(CoordinatesOverTime_mm);

 % Generate radii for the cylinders in the range
 radii = radii*ones(N,1);

 % Generate heights for the cylinders
 heights = heights*ones(N,1);


 clearvars -except stlcoords radii heights CoordinatesOverTime_mm CoolantOverTime spindleSpeed_rpm stl
grid_resolution_mm interpolations spiderMRR spiderPower spiderSEC q gcode steps Nmax stl stlcoords
coefficientsSEC coefficientsPower resolutions varMRR varSEC varPower
```

Get MRR for each timestep

```
 figure;

 %build bounding box for stock material
 xmin=min(stlcoords(:,1,:),[],"all");
 xmax=max(stlcoords(:,1,:),[],"all");
 ymin=min(stlcoords(:,2,:),[],"all");
 ymax=max(stlcoords(:,2,:),[],"all");
 zmin=min(stlcoords(:,3,:),[],"all");
 zmax=max(stlcoords(:,3,:),[],"all");
 minlength_mm=min([xmax-xmin,ymax-ymin,zmax-zmin]);

 % Create a 3D grid
 [X, Y, Z] = meshgrid(xmin:grid_resolution_mm:xmax, ...
                      ymin:grid_resolution_mm:ymax, ...
                      zmin:grid_resolution_mm:zmax);

 %Voxelise the STL:
 [voxel_mesh] = VOXELISE( xmin:grid_resolution_mm:xmax, ...
                      ymin:grid_resolution_mm:ymax, ...
                      zmin:grid_resolution_mm:zmax, ...
                      stl,'xyz');
 voxel_mesh=permute(voxel_mesh,[2,1,3]); %swap x and y
 drawMesh(X,Y,Z,voxel_mesh,'') %draw the voxel mesh

 %initialize array of volumes
 Volume=[sum(voxel_mesh(:)) * grid_resolution_mm^3];

 % Number of cylinders
 N = size(CoordinatesOverTime_mm, 1);

 %calculate Volume at each timestep
 for i=1:length(CoordinatesOverTime_mm)
     [Volume(i+1),voxel_mesh]=calculateCylinderVolumesAndDisplayMesh(CoordinatesOverTime_mm(1:i,:),
radii(1:i), heights(1:i),voxel_mesh,grid_resolution_mm,X,Y,Z);
 end
```

```
%Calculate volume change per timestep
dV=-(Volume(2:end)-Volume(1:end-1));

dt=(vertcat(CoordinatesOverTime_mm(1,4),CoordinatesOverTime_mm(2:end,4)))';
dt=dt.*60;

%calculate MRR per timestep
MRR_t=dV./dt;

%Visualize
[Volume(i+1),voxel_mesh]=calculateCylinderVolumesAndDisplayMesh(CoordinatesOverTime_mm(1:i,:),
radii(1:i), heights(1:i),voxel_mesh,grid_resolution_mm,X,Y,Z);
figure
drawMesh(X,Y,Z,voxel_mesh,'')
hold on
plot3(CoordinatesOverTime_mm(:,1),CoordinatesOverTime_mm(:,2),CoordinatesOverTime_mm(:,3))
hold off
clear X Y Z
```

Get Total Energy

```
%Get SEC and power
for i=1:length(MRR_t)
    SEC_all(i)=getSEC(coefficientsSEC,spindleSpeed_rpm(i),(MRR_t(i)));
    Power_all(i)=getPower(coefficientsPower,spindleSpeed_rpm(i),(MRR_t(i)));
end

%Non cutting energy change
nonCuttingPoints=(isinf(SEC_all)|isnan(SEC_all));
noncuttingPower=coefficientsPower(1)+coefficientsPower(2)*spindleSpeed_rpm(nonCuttingPoints); %double
check to see if true
dEidle=noncuttingPower.*dt(nonCuttingPoints)';
fprintf("Non-Cutting Energy Consumption %f J",sum(dEidle))

%Get Change in Energy Consumption for cutting times
dEcut=SEC_all(~nonCuttingPoints).*dV(~nonCuttingPoints);
fprintf("Cutting Energy Consumption %f J",sum(dEcut))

%get SEC, MRR, and Power of cuttng times
SEC_c=SEC_all(~nonCuttingPoints);
MRR_c=MRR_t(~nonCuttingPoints);
Power_c=Power_all(~nonCuttingPoints);

%Display Average MRR, SEC, and Power of cutting times
fprintf("Average SEC: %f",mean(SEC_all(~nonCuttingPoints)))
fprintf("Average MRR: %f",mean(MRR_t(~nonCuttingPoints)))

cuttingPower=mean(Power_all(~nonCuttingPoints));
fprintf("Average Power: %f",mean(Power_all(~nonCuttingPoints)))
fprintf("spindle speed" + ".: %f",spindleSpeed_rpm(20));

%MRR Power and SEC during stabilized cut
startindex=ceil(radii/164*length(SEC_c))+1;%lower bound where cut MRR begins to be stable
endindex=floor((164-2*radii)/164*length(SEC_c))-1; %upper bound for where MRR begins to be stable
MRR_s= MRR_c(startindex:endindex);
SEC_s= SEC_c(startindex:endindex);
Power_s= Power_c(startindex:endindex);

%Display Average MRR, SEC, and Power ignoring unstable points
fprintf("Average stabilized SEC: %f",mean(SEC_s))
fprintf("Average stabilized MRR: %f",mean(MRR_s))
fprintf("Average stabilized Power: %f",mean(Power_s))

%Get Change in Energy from Coolant Running
coolantRuntime=sum(dt'.*CoolantOverTime,"all")

%Get Sum of Energy Consumption
```

```
E=sum(dEcut)+sum(dEidle);
fprintf("Total Energy Consumption %f J",E)
clear figure
end
```

```
function [fitresultSEC,fitresultPower, gof] = createFit(MRR, n, Power)
%CREATEFIT(MRR,N,SEC)
%  Create a fit.
%
%  Data for 'untitled fit 1' fit:
%      X Input: MRR (mm3)
%      Y Input: n
%      Z Output: Power (kW)
%  Output:
%      fitresult : a fit object representing the fit.
%      gof : structure with goodness-of fit info.
%
%  See also FIT, CFIT, SFIT.

%  Auto-generated by MATLAB on 27-Oct-2024 16:02:56
%
% n = spindle seed rpm
% MRR = array of test MRR in mm3/s
% SEC = array list of specific cutting energies measured at MRR in J/mm3

%% Fit: 'untitled fit 1'.
[xData, yData, zData] = prepareSurfaceData( MRR, n, Power );

% Set up fittype and options.
ft = fittype( 'a+(b*n)+c*(n^d)*MRR', 'independent', {'MRR', 'n'}, 'dependent', 'SEC' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.Lower = [0 0 0 0];
opts.StartPoint = [1 1 1 1];

% Fit model to data.
[fitresultPower, gof] = fit( [xData, yData], zData, ft, opts );

% Plot fit with data.
figure( 'Name', 'Power vs MRR and Spindle Speed' );
h = plot( fitresultPower, [xData, yData], zData );
% Label axes
xlabel( 'MRR', 'Interpreter', 'none' );
ylabel( 'n', 'Interpreter', 'none' );
zlabel( 'Power', 'Interpreter', 'none' );


SEC=(Power.*1000)./MRR;

%% Fit: 'untitled fit 1'.
[xData, yData, zData] = prepareSurfaceData( MRR, n, SEC );

% Set up fittype and options.
ft = fittype( 'a*(n^b)+((c*n)/MRR)+d/MRR', 'independent', {'MRR', 'n'}, 'dependent', 'SEC' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.Lower = [0 0 0 0];
opts.StartPoint = [1 1 1 1];

% Fit model to data.
[fitresultSEC, gof] = fit( [xData, yData], zData, ft, opts );

% Plot fit with data.
figure( 'Name', 'SEC vs MRR and Spindle Speed' );
h = plot( fitresultSEC, [xData, yData], zData );
```

```
% Label axes
xlabel( 'MRR', 'Interpreter', 'none' );
ylabel( 'n', 'Interpreter', 'none' );
zlabel( 'SEC', 'Interpreter', 'none' );
grid on

end

function SEC = getSEC(coeff,n,MRR)
% SEC model from Zhou (2017)
% inputs:
% coeff = coefficients from createFit
% n = spindle speed (rpm)
% MRR = Material removal rate (mm3/s)
    SEC=coeff(1)*(n^coeff(2))+((coeff(3)*n)/MRR)+coeff(4)/MRR;
end

function Power = getPower(coeff,n,MRR)
% Power model from Zhou (2017)
% inputs:
% coeff = coefficients from createFit
% n = spindle speed (rpm)
% MRR = Material removal rate (mm3/s)
    Power=coeff(1)+(coeff(2)*n)+coeff(3)*(n^coeff(4))*MRR;
end

function G = GetGs(N,Table)
    G=split(Table{N,"G"},",");
end

function [Movement, Distance,Plane] = interpretGs(N,Table)
global Gmatrix
G=GetGs(N,Table);

MoveTypes = [00,01,02,03,04];
DistanceTypes=[90,91];
PlaneTypes=[17,18,19];

for i = 2:length(G) % parse line for movement and distance type ignoring A
    currentG = str2num(G(i));

    if any(MoveTypes==currentG)
        Movement = currentG;
    end

    if any(DistanceTypes==currentG)
        Distance = currentG;
    end

    if any(PlaneTypes==currentG)
        Plane = currentG;
    end

end

    if ~exist("Movement","var")
       if N==1
           Movement = Gmatrix(N,1);
       else
           Movement = Gmatrix(N-1,1);
       end
    end

    if ~exist("Distance","var")
       if N==1
           Distance = Gmatrix(N,2);
       else
         Distance = Gmatrix(N-1,2);
       end
```

```
    end

    if ~exist("Plane","var")
       if N==1
           Plane = Gmatrix(N,3);
       else
         Plane = Gmatrix(N-1,3);
       end
    end

end

function [NewCoordinates, len, time] = AbsLinMove(N,Table, coords, feed)
%absolute linear motion
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    X = Table{N,'X'};
    Y = Table{N,'Y'};
    Z = Table{N,'Z'};

    to = [X,Y,Z];
    for i=1:3
        if isnan(to(i))
            to(i)=from(i);
        end
    end
    NewCoordinates = to;
    len = norm(NewCoordinates-from);
    if isempty(feed)
        time=0;
        NewCoordinates=from;
    else
        time = len / feed;
    end

end

function [NewCoordinates, len, time] = RelLinMove(N,Table, coords, feed)
%relative linear motion
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    X = Table{N,'X'};
    Y = Table{N,'Y'};
    Z = Table{N,'Z'};

    to = [X,Y,Z];
    for i=1:3
        if isnan(to(i))
            to(i)=from(i);
        end
    end
    NewCoordinates = to;
    len = norm(NewCoordinates-from);
    if isempty(feed)
        time=0;
        NewCoordinates=from;
    else
        time = len / feed;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%arc motions can probably be collapsed into one function
function [NewCoordinates, len, time] = CCWAbsArcMoveCenter(N,Table, coords, feed)
%counterclockwise absolute arc motion center motion
global pitch
global radius
global nTurns
global angle
global angle1
global Center
```

```
global Gmatrix

from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = AbsLinMove(N,Table,coords,feed);
    I = Table{N,'I'};
    J = Table{N,'J'};
    K = Table{N,'K'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    offsets=[I,J,K];
    offsets(isnan(offsets))=0;

    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,offsets(1:2));
    clear EndPoint StartPoint
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(1),offsets(3)]);
    angle=2*pi-angle; % fix plane issue
    clear EndPoint StartPoint
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(2),offsets(3)]);
    clear EndPoint StartPoint
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end

function [NewCoordinates, len, time] = CWAbsArcMoveCenter(N,Table, coords, feed)
%clockwise relative arc motion center
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = AbsLinMove(N,Table,coords,feed);
    I = Table{N,'I'};
    J = Table{N,'J'};
    K = Table{N,'K'};
    P = Table{N,'P'};
    if isnan(P)
```

```
        P=1;
    end
    offsets=[I,J,K];
    offsets(isnan(offsets))=0;

    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,offsets(1:2));
    clear EndPoint StartPoint
    angle=2*pi-angle;
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(1),offsets(3)]);
    clear EndPoint StartPoint
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(2),offsets(3)]);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end

end

function [NewCoordinates, len, time] = CCWRelArcMoveCenter(N,Table, coords, feed)
%counterclockwise relative arc move center
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = RelLinMove(N,Table,coords,feed);
    I = Table{N,'I'};
    J = Table{N,'J'};
    K = Table{N,'K'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    offsets=[I,J,K];
    offsets(isnan(offsets))=0;

    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
```

```
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,offsets(1:2));
    clear EndPoint StartPoint
    radius = norm(offsets); %radius of circle
    height = abs(from(3)-NewCoordinates(3));
    nTurns = P;
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(1),offsets(3)]);
    angle=2*pi-angle; % fix plane issue
    clear EndPoint StartPoint
    radius = norm(offsets); %radius of circle
    height = abs(from(2)-NewCoordinates(2));
    nTurns = P;
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(2),offsets(3)]);
    clear EndPoint StartPoint
    radius = norm(offsets); %radius of circle
    height = abs(from(1)-NewCoordinates(1));
    nTurns = P;
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end

function [NewCoordinates, len, time] = CWRelArcMoveCenter(N,Table, coords, feed)
%clockwise relative arc motion center
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = RelLinMove(N,Table,coords,feed);
    I = Table{N,'I'};
    J = Table{N,'J'};
    K = Table{N,'K'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    offsets=[I,J,K];
    offsets(isnan(offsets))=0;

    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,offsets(1:2));
    clear EndPoint StartPoint
    angle=2*pi-angle;
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
```

```
    time = len / feed;

    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(1),offsets(3)]);
    clear EndPoint StartPoint
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectors(StartPoint,EndPoint,[offsets(2),offsets(3)]);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    radius=norm(offsets); %radius of circle
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle)/2);
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end

function [NewCoordinates, len, time] = CWAbsArcMoveRadius(N,Table, coords, feed)
%clockwise relative arc motion radius
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = AbsLinMove(N,Table,coords,feed);
    radius = Table{N,'R'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;

    elseif Gmatrix(N,3)==19
```

```
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end

function [NewCoordinates, len, time] = CCWAbsArcMoveRadius(N,Table, coords, feed)
%counterclockwise relative arc motion radius
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = AbsLinMove(N,Table,coords,feed);
    radius = Table{N,'R'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    angle=2*pi-angle; % fix plane issue
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end


function [NewCoordinates, len, time] = CWRelArcMoveRadius(N,Table, coords, feed)
%clockwise relative arc motion radius
global pitch
global radius
global nTurns
```

```
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = RelLinMove(N,Table,coords,feed);
    radius = Table{N,'R'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end
    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle, Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    nTurns = P;
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle, Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle, Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    angle=2*pi-angle;
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end

function [NewCoordinates, len, time] = CCWRelArcMoveRadius(N,Table, coords, feed)
%counterclockwise relative arc motion radius
global pitch
global radius
global nTurns
global angle
global angle1
global Center
global Gmatrix
from = [coords{N,"X1"},coords{N,"Y1"},coords{N,"Z1"}];
    [NewCoordinates,~,~] = RelLinMove(N,Table,coords,feed);
    radius = Table{N,'R'};
    P = Table{N,'P'};
    if isnan(P)
        P=1;
    end

    if Gmatrix(N,3)==17
    StartPoint=[from(1),from(2)];
    EndPoint=[NewCoordinates(1),NewCoordinates(2)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
```

```
    height = abs(from(3)-NewCoordinates(3));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==18
    StartPoint=[from(1),from(3)];
    EndPoint=[NewCoordinates(1),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    angle=2*pi-angle; % fix plane issue
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(2)-NewCoordinates(2));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    elseif Gmatrix(N,3)==19
    StartPoint=[from(2),from(3)];
    EndPoint=[NewCoordinates(2),NewCoordinates(3)];
    [angle1,angle,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,radius);
    clear EndPoint StartPoint
    nTurns = P;
    height = abs(from(1)-NewCoordinates(1));
    pitch = height/((2*pi*(nTurns-1)+angle));
    len=nTurns*sqrt((2*pi*radius)^2+pitch^2);
    time = len / feed;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [NewCoordinates,len,time,spindle] = DeterminePath(N,Table,coords) % goes to line N and reads what path type it is, spits out end coordinates, path length, and time for line
    global Gmatrix
    global CurrentMove
    global RapidFeed
    feed = Table{find(~isnan(Table{1:N,"F"}),1,"last"),"F"};
    spindle = Table{find(~isnan(Table{1:N,"S"}),1,"last"),"S"};
    if Gmatrix(N,2)==90 %if absolute move

         if Gmatrix(N,1)==0 %if linear rapid
            CurrentMove="Linear";

            [NewCoordinates,len,time] = AbsLinMove(N,Table,coords,RapidFeed);


         elseif Gmatrix(N,1)==1 %if linear move
            CurrentMove="Linear";

            [NewCoordinates, len, time] = AbsLinMove(N,Table, coords, feed);


         elseif Gmatrix(N,1)==2 %if clockwise move
            CurrentMove="CW Arc";
            if ~isnan(Table{N,"R"}) %if radius format
                [NewCoordinates, len, time] = CWAbsArcMoveRadius(N,Table, coords, feed);

            else %if center format
                [NewCoordinates, len, time] = CWAbsArcMoveCenter(N,Table, coords, feed);
            end


         elseif Gmatrix(N,1)==3 %if counter clockwise move
             CurrentMove="CCW Arc";
             if ~isnan(Table{N,"R"}) %if radius format
                [NewCoordinates, len, time] = CCWAbsArcMoveRadius(N,Table, coords, feed);

            else %if center format
                [NewCoordinates, len, time] = CCWAbsArcMoveCenter(N,Table, coords, feed);
             end
```

```
            else
                NewCoordinates=table2array(coords);
                len=0;
                time=0;
                disp("no path")
            end

        elseif Gmatrix(N,2)==91 %if relative move

            if Gmatrix(N,1)==0 %if linear rapid
               CurrentMove="Linear";
               [NewCoordinates,len,time] = RelLinMove(N,Table,coords,RapidFeed);


            elseif Gmatrix(N,1)==1 %if linear move
               CurrentMove="Linear";
               [NewCoordinates, len, time] = RelLinMove(N,Table, coords, feed);


            elseif Gmatrix(N,1)==2 %if clockwise move
               CurrentMove="CW Arc";
               if ~isnan(Table{N,"R"}) %if radius format
                   [NewCoordinates, len, time] = CWRelArcMoveRadius(N,Table, coords, feed);

               else %if center format
                   [NewCoordinates, len, time] = CWRelArcMoveCenter(N,Table, coords, feed);
               end


            elseif Gmatrix(N,1)==3 %if counter clockwise move
                CurrentMove="CCW Arc";
                if ~isnan(Table{N,"R"}) %if radius format
                   [NewCoordinates, len, time] = CCWRelArcMoveRadius(N,Table, coords, feed);

               else %if center format
                   [NewCoordinates, len, time] = CCWRelArcMoveCenter(N,Table, coords, feed);
                end

            else
                disp("Error in Determine path")
            end


        end
    end

    function [theta_0,theta,Center]= findAngleBtwVectors(StartPoint,EndPoint,Offset)
    %Center=center point
    CenterToStart=-Offset;
    CenterToEnd=EndPoint-(StartPoint+Offset);
    Center=StartPoint+Offset;
    theta_0=atan2(CenterToStart(2),CenterToStart(1));

    theta_1=atan2(CenterToEnd(2),CenterToEnd(1));
    if theta_1<0
        theta_1=theta_1+2*pi;
    end
    if theta_0<0
        theta_0=theta_0+2*pi;
    end
    theta=theta_1-theta_0;
    if theta<0
        theta=theta+2*pi;
    end
    end

    function [theta_0,theta,Center] = findAngleBtwVectorsRadius(StartPoint,EndPoint,Radius)
[xout,yout]=circcirc(StartPoint(1),StartPoint(2),abs(Radius),EndPoint(1),EndPoint(2),abs(Radius)); %find
centerpoint
```

```
 if any(isnan(xout))

[xout,yout]=circcirc(StartPoint(1),StartPoint(2),abs(Radius)+eps,EndPoint(1),EndPoint(2),abs(Radius)+eps)
; %find centerpoint with small increase in vectors
 end
 Center1=[xout(1),yout(1)];
 Center2=[xout(2),yout(2)];

 CenterToStart=StartPoint-Center1;
 CenterToEnd=EndPoint-Center1;
 theta_0=atan2(CenterToStart(2),CenterToStart(1));
 theta_1=atan2(CenterToEnd(2),CenterToEnd(1));
 if theta_1<=0
     theta_1=theta_1+2*pi;
 end
 if theta_0<0
     theta_0=theta_0+2*pi;
 end
 theta=theta_1-theta_0;
 if theta<0
     theta=theta+2*pi;
 end
 Center=Center1;
 %choose center based on which has the smaller change in angle (or larger if
 %negative Radius
 if theta<pi && Radius>0
     CenterToStart=StartPoint-Center2;
     CenterToEnd=EndPoint-Center2;
     theta_0=atan2(CenterToStart(2),CenterToStart(1));
     theta_1=atan2(CenterToEnd(2),CenterToEnd(1));
     if theta_1<0
         theta_1=theta_1+2*pi;
     end
     if theta_0<0
      theta_0=theta_0+2*pi;
     end
     theta=theta_1-theta_0;
     if theta<0
      theta=theta+2*pi;
      Center=Center2;
 end
 end
 end

 function [total_volume,voxel_mesh] = calculateCylinderVolumesAndDisplayMesh(positions, radii,
heights,voxel_mesh,grid_resolution,X,Y,Z)
     % Function to calculate the total volume of cylinders considering overlaps
     % and display a 3D mesh of the overlapping regions.
     % Inputs:
     %   positions - Nx3 matrix of (x, y, z) coordinates for the cylinders
     %   radii - Nx1 array of radii for each cylinder
     %   heights - Nx1 arfindAngleray of heights for each cylinder
     % Output:
     %   total_volume - The total volume of all cylinders, considering overlaps

     % Number of cylinders
     N = size(positions, 1);

         % Check each cylinder to see if grid points fall within its volume

         % Starter Position Cylinder parameters
         if N==1
         Center(1,:)=positions(N,:);
         else
         Center(1,:)=positions(N-1,:);
         end

         % Final PositionCylinder parameters
         Center(2,:)=positions(N,:);
```

```
        r = radii(N);
        h = heights(N);

        % Check which points are inside the current cylinder
        inside_cylinder = ((X - Center(2,1)).^2 + (Y - Center(2,2)).^2 <= r^2 & ...
                           Z >= Center(2,3) & Z <= Center(2,3) + h);
        % Update the overall logical array
        voxel_mesh = voxel_mesh & ~inside_cylinder;
        clear inside_cylinder

 % Calculate the direction vector from Cylinder 1 to Cylinder 2
 vec = [Center(2,1) - Center(1,1), Center(2,2) - Center(1,2), Center(2,3) - Center(1,3)];
 dist = norm(vec);           % Distance between cylinder bases
 vec = vec / dist;           % Unit direction vector from Cylinder 1 to Cylinder 2

 % find which circle is base
 Base= find(Center(:,3)==min(Center(1,3),Center(2,3)),1,"first");
 Top= find(Center(:,3)==max(Center(1,3),Center(2,3)),1,"last"); %index of cylinder that will be the top
for oblique cylinder

 % Calculate angle of cylinder projection
 thetaxz=atan(vec(1)/vec(3));
 thetayz=atan(vec(2)/vec(3));

 %slope of projection to xy
 M = (Center(2,2) - Center(1,2))/(Center(2,1) - Center(1,1));

 top_cylinder_mask = (((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz)).^2+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz)).^2<=r^2 &...%define top of oblique rectangle
                     Z>=Center(Base,3)+h & Z<=Center(Top,3)+h);
 voxel_mesh = voxel_mesh & ~top_cylinder_mask;
 clear top_cylinder_mask
 bottom_cylinder_mask = (((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz)).^2+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz)).^2<=r^2 &...%define top of oblique rectangle
                     Z>=Center(Base,3) & Z<=Center(Top,3));
 voxel_mesh = voxel_mesh & ~bottom_cylinder_mask;
 clear bottom_cylinder_mask


 if sign(M)==sign(vec(1))
 rectangle_mask=(abs(Y-Center(Base,2)-(M.*(X-Center(Base,1))))<=r*sqrt(1+M^2) &...%define xy movement
bound
                 X>=min(Center(1,1),Center(2,1))-r & X<=max(Center(1,1),Center(2,1))+r &... %side bounds
                 Y>=min(Center(1,2),Center(2,2))-r & Y<=max(Center(1,2),Center(2,2))+r &...
                 ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound of cylinder
                 ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0 &... %top bound of rect
                 Z>=Center(Base,3) & Z<=Center(Base,3)+h &... % full top and bottom bound
                 Y-Center(1,2)>=(-1/M)*(X-Center(1,1)) &...
                 Y-Center(2,2)<=(-1/M)*(X-Center(2,1))); %end bounds
                 voxel_mesh = voxel_mesh & ~rectangle_mask;

 elseif sign(M)==sign(vec(1))*-1
        rectangle_mask=(abs(Y-Center(Base,2)-(M.*(X-Center(Base,1))))<=r*sqrt(1+M^2) &...%define xy
movement bound
                 X>=min(Center(1,1),Center(2,1))-r & X<=max(Center(1,1),Center(2,1))+r &...
                 Y>=min(Center(1,2),Center(2,2))-r & Y<=max(Center(1,2),Center(2,2))+r &...
                 ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound
                 ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0 &... %top bound
                 Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                 Y-Center(1,2)<=(-1/M)*(X-Center(1,1)) &...
                 Y-Center(2,2)>=(-1/M)*(X-Center(2,1)));
                 voxel_mesh = voxel_mesh & ~rectangle_mask;

 elseif M==0 %if horizontal slope
     if sign(vec(1))==1
```

```
        if Center(1,3)~=Center(2,3)
            rectangle_mask=(abs(Y-Center(Base,2))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                X>=(Center(1,1)) &...
                X<=(Center(2,1)) &...
                ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound
                ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0); %top bound
                voxel_mesh = voxel_mesh & ~rectangle_mask;
        else
            rectangle_mask=((abs(Y-Center(Base,2))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                X>=(Center(1,1)) &...
                X<=(Center(2,1))));
                voxel_mesh = voxel_mesh & ~rectangle_mask;
        end
    else
        if Center(1,3)~=Center(2,3)
             rectangle_mask=(abs(Y-Center(Base,2))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                X<=(Center(1,1)) &...
                X>=(Center(2,1)) &...
                ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound
                ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0); %top bound
                voxel_mesh = voxel_mesh & ~rectangle_mask;
        else
             rectangle_mask=(abs(Y-Center(Base,2))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                X<=(Center(1,1)) &...
                X>=(Center(2,1)));
                voxel_mesh = voxel_mesh & ~rectangle_mask;
        end
    end

 elseif isinf(M) %if vertical slope
        if sign(vec(2))==1
            if Center(1,3)~=Center(2,3)
            rectangle_mask=(abs(X-Center(Base,1))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                Y>=(Center(1,2)) &...
                Y<=(Center(2,2))&...
                ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound
                ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0); %top bound
                voxel_mesh = voxel_mesh & ~rectangle_mask;
            else
                rectangle_mask=(abs(X-Center(Base,1))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                Y>=(Center(1,2)) &...
                Y<=(Center(2,2)));
                voxel_mesh = voxel_mesh & ~rectangle_mask;
            end
        else
            if Center(1,3)~=Center(2,3)
             rectangle_mask=(abs(X-Center(Base,1))<=r &...%define xy movement bound
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                Y<=(Center(1,2)) &...
                Y>=(Center(2,2)) &...%;
                ((X-Center(Base,1))-(Z-Center(Base,3)).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)).*tan(thetayz))<=0 &... %bottom bound
                ((X-Center(Base,1))-(Z-Center(Base,3)-h).*tan(thetaxz))+((Y-Center(Base,2))-(Z-
Center(Base,3)-h).*tan(thetayz))>=0); %top bound
                voxel_mesh = voxel_mesh & ~rectangle_mask;
            else
                rectangle_mask=(abs(X-Center(Base,1))<=r &...%define xy movement bound
```

```
                Z>=Center(Base,3) & Z<=Center(Base,3)+h &...
                Y<=(Center(1,2)) &...
                Y>=(Center(2,2)));
                voxel_mesh = voxel_mesh & ~rectangle_mask;
            end
        end
end
   clear rectangle_mask



    % Estimate the total volume based on the number of points inside cylinders
    volume_per_point = grid_resolution^3;
    total_volume = sum(voxel_mesh,"all") * volume_per_point;

end

function drawMesh(X,Y,Z,logical_array,i)
    if i=="invert"
        logical_array=not(logical_array);
    end

    % Plot the mesh of the overlapping region
    figure;
    % Use isosurface to create the 3D surface plot
    p = patch(isosurface(X, Y, Z, logical_array, 0.5),'FaceColor',[.5 .5 .5],'EdgeColor','none');
    isonormals(X, Y, Z, logical_array, p);

    % Enhance the plot with lighting and perspective
    camlight;
    lighting none;
    axis equal;
    xlabel('X [mm]');
    ylabel('Y [mm]');
    zlabel('Z [mm]');
    title('3D Mesh of Material');
    grid on;
    view(3);
end
```

Bibliography

[1] 2016, *International Energy Outlook 2016*, U.S. Energy Information Administration (EIA). [Online]. Available: https://www.eia.gov/outlooks/ieo/pdf/0484(2016).pdf.

[2] Gutowski, T., Dahmus, J., and Thiriez, A., 2006, "Electrical Energy Requirements for Manufacturing Processes ." [Online]. Available: https://web.mit.edu/2.813/www/readings/Gutowski-CIRP.pdf.

[3] Li, L., Yan, J., and Xing, Z., 2013, "Energy Requirements Evaluation of Milling Machines Based on Thermal Equilibrium and Empirical Modelling," Journal of Cleaner Production, **52**, pp. 113–121. https://doi.org/10.1016/j.jclepro.2013.02.039.

[4] Li, W., and Kara, S., 2011, "An Empirical Model for Predicting Energy Consumption of Manufacturing Processes: A Case of Turning Process," Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, **225**(9), pp. 1636–1646. https://doi.org/10.1177/2041297511398541.

[5] Kordonowy, D., 2022, "A Power Assessment of Machining Tools," B.S. Thesis, Massachusetts Institute of Technology. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/31108/52949299-MIT.pdf?sequence=2&isAllowed=y.

[6] Shin, S.-J., Woo, J., and Rachuri, S., 2017, "Energy Efficiency of Milling Machining: Component Modeling and Online Optimization of Cutting Parameters," Journal of Cleaner Production, **161**, pp. 12–29. https://doi.org/10.1016/j.jclepro.2017.05.013.

[7] Pantazis, D., Goodall, P., Pease, S. G., Conway, P., and West, A., 2023, "Predicting Electrical Power Consumption of End Milling Using a Virtual Machining Energy Toolkit (V_MET)," Computers in Industry, **150**, p. 103943. https://doi.org/10.1016/j.compind.2023.103943.

[8] Aitkenhead, A., 2013, "Polygon Mesh Voxelisation," MathWorks. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/27390-mesh-voxelisation.

[9] Zhou, L., Li, J., Li, F., Xu, X., Wang, L., Wang, G., and Kong, L., 2017, "An Improved Cutting Power Model of Machine Tools in Milling Process," Int J Adv Manuf Technol, **91**(5–8), pp. 2383–2400. https://doi.org/10.1007/s00170-016-9929-x.

[10] "FSWizard Machinist Calculator." [Online]. Available: https://app.fswizard.com. [Accessed: 22-May-2025].