MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

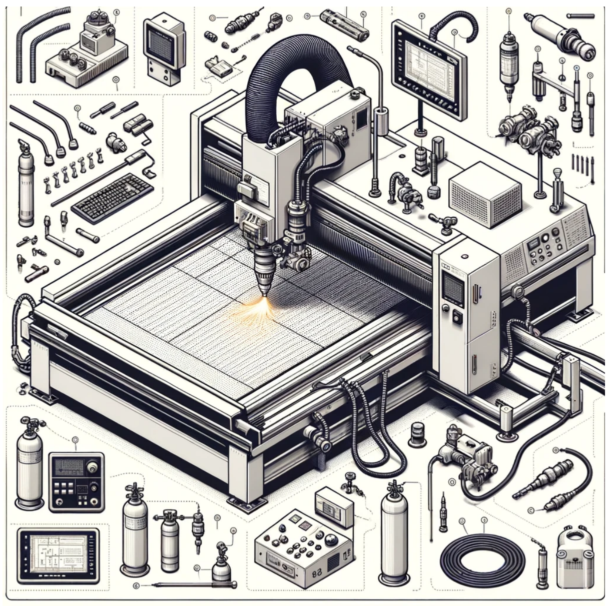
Development of a CNC Flame Cutting Machine Simulator

Student: Urdea Mara Cristina

Group: 30432

Coordinator: Assoc. Prof. Lia-Anca Hangan

January 2024



Contents

1 Introduction

The goal of this project is to develop a CNC [1] Flame Cutting Machine Simulator, a software program that reads a file containing a cutting path and generates commands to simulate the movement of a cutting head along the `X` and `Y` axes. The project will offer a valuable tool for testing and optimizing cutting paths before the actual execution, thus reducing material wastage and improving cutting accuracy.

2 Project Overview

3 Objectives

- Develop a user-friendly CNC Flame Cutting Machine Simulator.
- Provide a graphical user interface for user interaction and customization.
- Simulate the movement of the cutting head on a computer screen.
- Read input files containing cutting path information.
- Provide export functionality for generated CNC G-code.
- Allow users to analyze, visualize and adjust cutting paths.

4 Plan

1st Meeting – Project theme

- Choose the project theme.

2nd Meeting – Bibliographic study and project planning

- Define the project objectives and scope.
- Write a project plan.
- Identify key stakeholders and gather their requirements.
- Analyze the CNC Flame Cutting Machine.
- Find documentation sources.

3rd Meeting – Skeleton of the application

- Identify essential user interface requirements.
- Make a structure of the project.
- Familiarize with required languages (G-code, Python etc.).
- Start the implementation (skeleton).

[1] Computer Numerical Control

4th Meeting - Development

- Development work.
- Review the progress.

5th Meeting - Development

- Development work.
- Start writing the documentation.
- Review the progress.

6th Meeting – Final improvement

- Final feedback.
- Define the final shape of the project.

5 Bibliographic Study

For a better understanding of the experiment, Figure.1 represents the flame cutting machine while cutting a piece of metal. We can observe the oxygen along with the preheated gases that will perform the cutting.

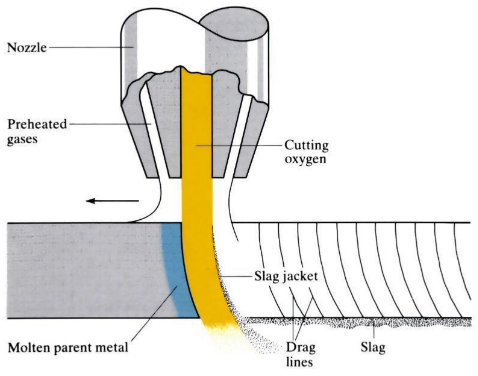


Figure 1: Flame Cutting Scheme

The current project focuses on the algorithm that helps the machine move according to the user-selected coordinates and simulate this movement on the computer's screen.

The working process consists of five important steps:

- Creating the design of the machine and the simulation environment using a GUI.
- Retrieving the movement coordinates from a text file.
- Transmitting these coordinates to the machine.
- Simulating the movement based on the coordinates.
- Generating the G-code for the design, that can be used to set up the actual CNC machine.

The user should be allowed to draw lines by inputting the `X` and `Y` coordinates, the start and end of the line. Also, the arcs will be drawn by inputting the start and end of the arc and the radius.

6 Analysis

6.1 G-Code

G-code, short for Geometric Code, is a standard programming language used to control computer numerical control (CNC) machines and 3D printers. It serves as the bridge between computer-aided design (CAD) software and the physical movements of CNC machines, dictating precise instructions on how the machine should operate. G-code is a fundamental component of CNC machining and plays a pivotal role in manufacturing, automation, and various other industries where precision is essential.

G-code is the language that CNC machines understand. It consists of a series of alphanumeric commands and parameters that instruct the machine how to move, what tools to use, and other crucial operational details. These commands control elements such as the tool's position, speed, direction, and tool changes.

Instructions are structured with a letter (often a 'G' for "geometry" or other letters denoting specific functions) followed by a number. For example, `'G00'` signifies rapid positioning, `'G01'` specifies linear interpolation (movement), and `'G02'` or `'G03'` instructs the machine to move in arcs. The instruction is often followed by numerical parameters, which provide specific details like coordinates, feed rates, and spindle speeds.

The most common coordinate system used in CNC machining is the Cartesian coordinate system, where `X`, `Y`, and `Z` axes represent three-dimensional space. These axes dictate the tool's position and movement in relation to the workpiece.

A line of G-code has the following structure:

```
G## X## Y## Z## F##
```

- First is the G-code command and in this case that's the `G01` which means "move in straight line to a specific position".[2]
- We declare the position or the coordinates with the X, Y and Z values.
- Lastly, with the F value we set the feed rate, or the speed at which the move will be executed.

To wrap up, the line `G01 X247 Y11 Z-1 F40` tells the CNC machine to move in a straight line from its current position to the coordinates `X247`, `Y11` and `Z-1` with speed of 40 mm/min.

6.2 G-Code from Cartesian coordinates

The algorithm for generating G-Code from Cartesian coordinates will take as inputs the `X` and `Y`, the feed rate (the rate at which the CNC machine should move) and the name of the G-Code file to be generated (for example ``output.gcode''). It starts by creating

[2]See Annex 11

an empty list to store the G-Code commands (`'G21'`: sets the units to millimeters, `'G90'`: sets the positioning to mode absolute, `'G1 F{feed_rate}'`: sets the feed rate for movement).

Then, it iterates through the provided Cartesian coordinates and generates G-Code for each point. For each coordinate pair `(X, Y)`, it appends a G-Code command to move the CNC machine to that position using `'G1 Xx Yy'`.

Optionally, the program can add a G-Code command to return the CNC machine to the origin `(X=0, Y=0)` using `'G0 X0 Y0'`. Then the algorithm opens a file in write mode and writes the generates G-Code commands to that file.

6.3 Line Generation

The Line Generation feature in the application is built upon the theoretical foundation of Bresenham's line algorithm. This algorithm is a cornerstone in computer graphics for drawing straight lines on raster displays. It operates by iteratively determining which pixel should be plotted to form the closest approximation to a straight line between two given points.

The algorithm's efficiency lies in its use of integer arithmetic rather than floating-point computations. It calculates the pixels needed to draw a line by moving across the x-axis in unit intervals and at each step, selecting the nearest pixel to the theoretical line. This selection is based on the error term that accumulates the divergence of the rasterized line from the ideal line.

Bresenham's algorithm is preferred in raster graphics environments due to its simplicity and the speed with which it can be executed, which is particularly valuable in real-time applications.

6.4 Circle/ Circle-arc Generation

The Circle and Arc Generation feature is grounded in the mathematical principles of circle drawing in raster graphics. A common approach to circle generation is the Midpoint Circle Algorithm, an extension of Bresenham's algorithm.

It efficiently draws circles by calculating the next pixel along the circumference from the current pixel, utilizing the circle's symmetry and the midpoint decision parameter. This parameter helps in deciding whether the next pixel to plot is either directly East or South-East of the current pixel. The algorithm effectively reduces the computational complexity by exploiting the circle's octant symmetry, thereby calculating the pixels for only one-eighth of the circle and replicating these for the remaining parts. For arc generation, similar principles are applied, but the plotting is confined to the arc segment specified by the start and end angles.

This method is advantageous for its low computational overhead and high accuracy in drawing circular paths, making it ideal for applications requiring geometric precision in rendering circular and arc-shaped graphical elements.

7 Design

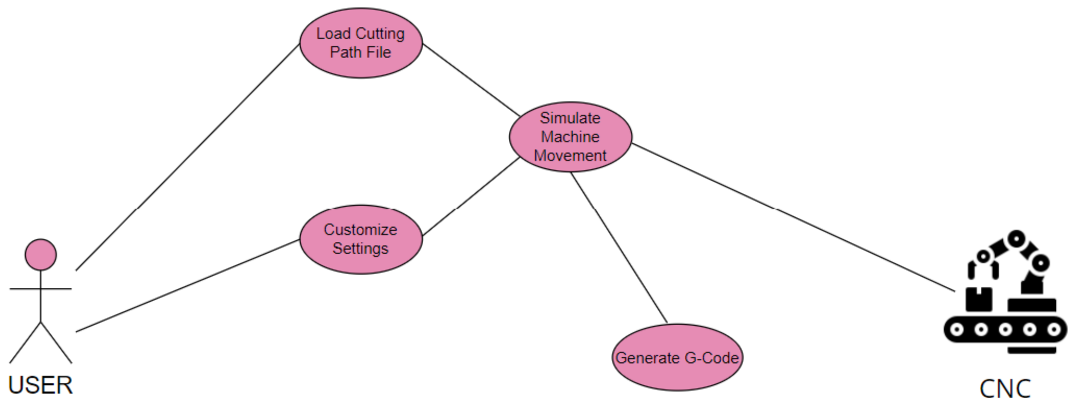


Figure 2: Use Case Diagram

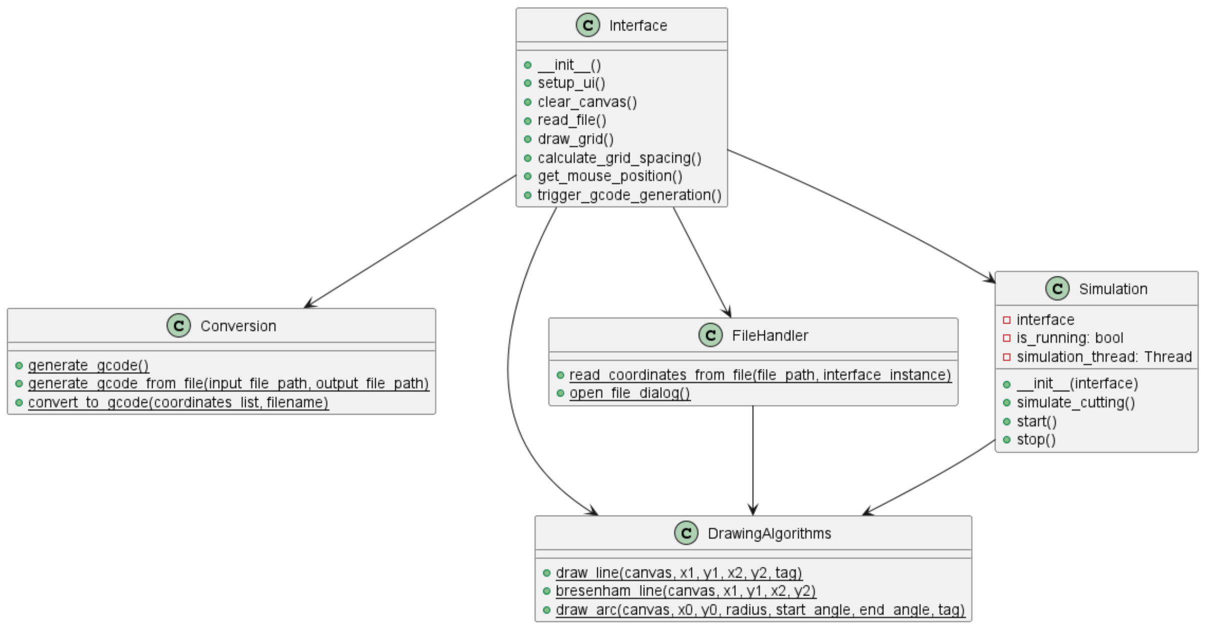

Interface
__init__()
setup_ui()
clear_canvas()
read_file()
draw_grid()
calculate_grid_spacing()
get_mouse_position()
trigger_gcode_generation()
Conversion
generate_gcode()
generate_gcode_from_file(input_file_path, output_file_path)
convert_to_gcode(coordinates_list, filename)
FileHandler
read_coordinates_from_file(file_path, interface_instance)
open_file_dialog()
Simulation
interface
is_running: bool
simulation_thread: Thread
__init__(interface)
simulate_cutting()
start()
stop()
DrawingAlgorithms
draw_line(canvas, x1, y1, x2, y2, tag)
bresenham_line(canvas, x1, y1, x2, y2)
draw_arc(canvas, x0, y0, radius, start_angle, end_angle, tag)

Figure 3: Unified Modeling Language (UML) Diagram

8 Implementation

This project is structured around several key Python classes, each handling distinct aspects of the application's functionality.

The Interface class forms the core of the user interface, managing UI interactions and canvas operations. It utilizes DrawingAlgorithms, a collection of static methods, for rendering geometric shapes and lines based on user inputs or file data.

The Conversion class is dedicated to generating G-code from drawing coordinates, enabling the translation of drawings into machine instructions. File operations, including reading coordinates and opening files, are managed by the FileHandler class, which interacts with the file system and processes input files.

Lastly, the Simulation class controls the simulation of the cutting process, offering start and stop capabilities while managing the simulation's runtime state. These classes work in concert to provide a comprehensive and interactive experience, from drawing and file handling to G-code generation and simulation of cutting paths.

9 Testing and Validation

Manual tests are performed to validate the user interface and the overall user experience. This includes testing the responsiveness of the UI, the accuracy of drawings on the canvas, and the correct execution of G-code generation and simulation functionalities.

The simulation is not perfect. Beause of some code and algorithm issues some shapes can not be displayed properly. These problems will be resolved in further development.

10 Conclusion

This project represents a significant step forward in integrating graphical user interfaces with automated machine control through G-code generation.

This user-friendly application bridges the gap between digital drawings and CNC machine operations.

Looking ahead, there are several avenues for further development. Enhancements such as support for more complex drawing algorithms, integration with additional file formats, and advanced simulation features would further increase the application's utility.

11 Bibliography

References

[1] "G-code Explained | List of Most Important G-code Commands", [Online]. Available: `https://howtomechatronics.com/tutorials/g-code-explained-list-of-most-important-g-code-commands/`

[2] "Learn", [Online]. Available: `https://www.learnpython.org/`

[3] "What Is CNC Machining? An Overview of the CNC Machining Process", [Online]. Available: `https://astromachineworks.com/what-is-cnc-machining/`

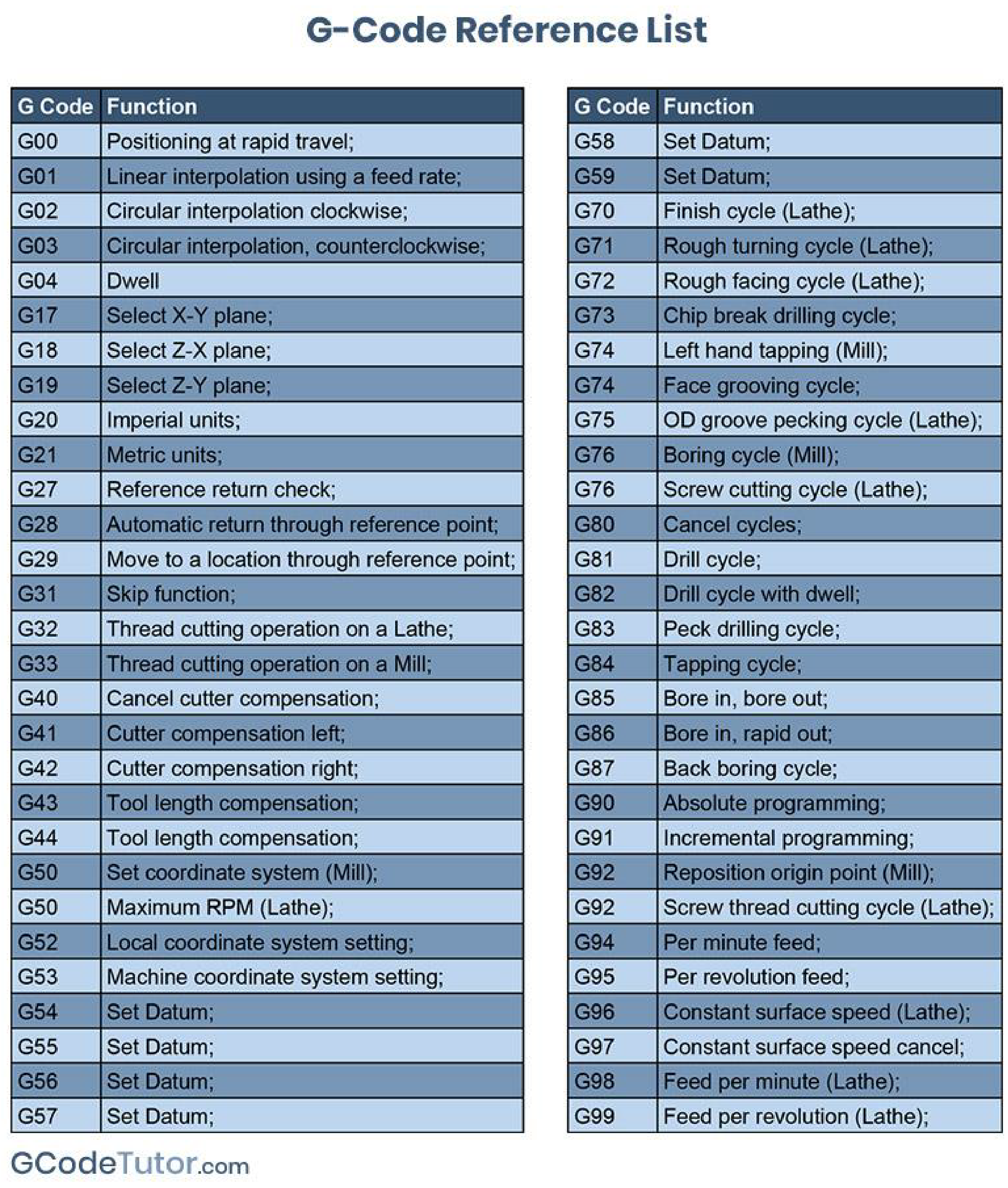
[4] "Bresenham's circle drawing algorithm", [Online]. Available: `https://www.geeksforgeeks.org/bresenhams-circle-drawing-algorithm/`

[5] "Flame cutting", [Online]. Available: `https://www.open.edu/openlearn/science-maths-technology/engineering-technology/manupedia/flame-cutting`

[6] "CNC G-Code", [Online]. Available: https://gcodetutor.com/cnc-machine-training/cnc-g-codes.html

[7] "Bresenham's Line Generation Algorithm", [Online]. Available: https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

Annex



G-Code Reference List

G Code	Function
G00	Positioning at rapid travel;
G01	Linear interpolation using a feed rate;
G02	Circular interpolation clockwise;
G03	Circular interpolation, counterclockwise;
G04	Dwell
G17	Select X-Y plane;
G18	Select Z-X plane;
G19	Select Z-Y plane;
G20	Imperial units;
G21	Metric units;
G27	Reference return check;
G28	Automatic return through reference point;
G29	Move to a location through reference point;
G31	Skip function;
G32	Thread cutting operation on a Lathe;
G33	Thread cutting operation on a Mill;
G40	Cancel cutter compensation;
G41	Cutter compensation left;
G42	Cutter compensation right;
G43	Tool length compensation;
G44	Tool length compensation;
G50	Set coordinate system (Mill);
G50	Maximum RPM (Lathe);
G52	Local coordinate system setting;
G53	Machine coordinate system setting;
G54	Set Datum;
G55	Set Datum;
G56	Set Datum;
G57	Set Datum;
G58	Set Datum;
G59	Set Datum;
G70	Finish cycle (Lathe);
G71	Rough turning cycle (Lathe);
G72	Rough facing cycle (Lathe);
G73	Chip break drilling cycle;
G74	Left hand tapping (Mill);
G74	Face grooving cycle;
G75	OD groove pecking cycle (Lathe);
G76	Boring cycle (Mill);
G76	Screw cutting cycle (Lathe);
G80	Cancel cycles;
G81	Drill cycle;
G82	Drill cycle with dwell;
G83	Peck drilling cycle;
G84	Tapping cycle;
G85	Bore in, bore out;
G86	Bore in, rapid out;
G87	Back boring cycle;
G90	Absolute programming;
G91	Incremental programming;
G92	Reposition origin point (Mill);
G92	Screw thread cutting cycle (Lathe);
G94	Per minute feed;
G95	Per revolution feed;
G96	Constant surface speed (Lathe);
G97	Constant surface speed cancel;
G98	Feed per minute (Lathe);
G99	Feed per revolution (Lathe);

GCodeTutor.com