Cutting Machine Simulation

Radu Stefanescu
Group 30434

Contents

1 Introduction

Introducing the CNC Simulation Program – a dynamic visual tool designed to bring the intricate processes of CNC machining into a clear and interactive digital environment.

CNC, or Computer Numerical Control, technology has revolutionized modern manufacturing by automating cutting and milling processes. These machines use digital instructions derived from a CAD (Computer-Aided Design) file. The versatility of CNC machines allows for high-precision production of a wide range of materials, from metals to plastics, enabling intricate designs and consistent quality. Their application spans numerous industries, including automotive, aerospace, and electronics, signifying their critical role in advanced manufacturing.

Figure 1: CNC Machine

CNC machines are marvels of modern engineering, capable of cutting and shaping materials with incredible precision based on programmed instructions. However, understanding and visualizing the complex paths involved in CNC machining can be a challenge, especially for those new to the field or those looking to troubleshoot and optimize existing designs.

That's where the simulation program steps in. With a focus on accessibility and user experience, this program serves as an essential educational and planning resource. By providing a simulated environment, users can observe the movement of the cutting head as it follows the programmed path, making the invisible visible and the complex simple.

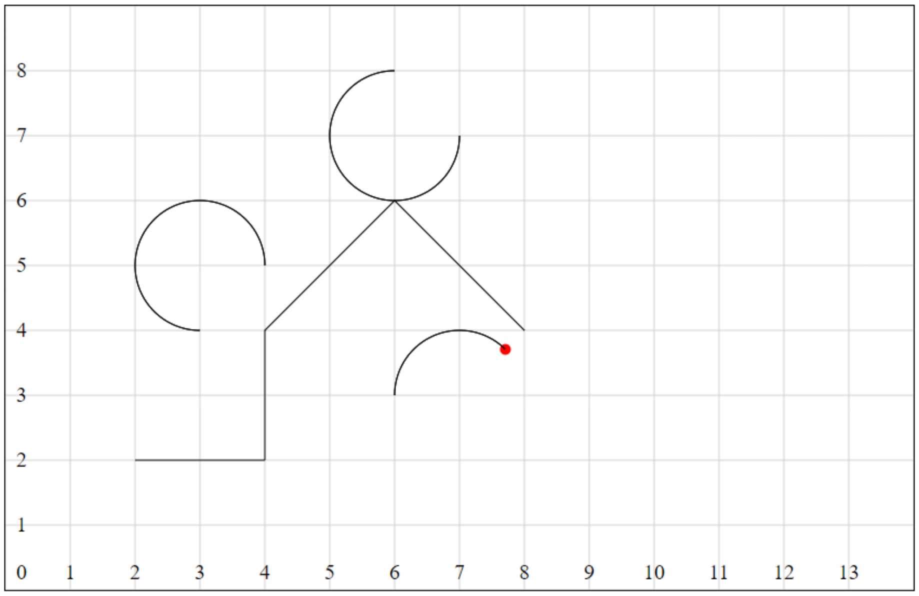
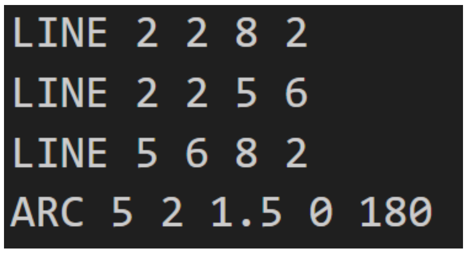


Figure 2: CNC program example

Whether you're an educator, student, engineer, or hobbyist, the CNC simulation program offers a valuable opportunity to see the potential of your designs come to life on screen before a single piece of material is cut. This not only saves time and resources but also empowers users with a deeper understanding of CNC machining dynamics, fostering innovation and enhancing the precision of your projects.

2 Analysis

First and foremost, we have to define a set of commands for the CNC machine that enables it to draw lines and arcs. These commands have to be simple and concise because they have to be written with ease by the people who will operate the cutting machine program.



```
LINE 2 2 8 2
LINE 2 2 5 6
LINE 5 6 8 2
ARC 5 2 1.5 0 180
```

Figure 3: CNC Machine commands

The design of the command set for the CNC machine focuses on simplicity and precision:

- **Line commands**: Structured to include coordinates for the starting point followed by the endpoint, enabling the machine to draw straight lines between these points.
- **Arc commands**: More intricate, requiring the coordinates of the circle's center, the radius, and then the start and end angles. These parameters facilitate the drawing of arcs in a counterclockwise direction.

This systematic approach ensures that both lines and arcs are executed with high precision, essential for the accurate rendering of complex designs.

The efficacy of our CNC Simulation Program hinges on its ability to accurately render animations based on the sequence and specifics of the input commands. It is crucial that the program interprets both line and arc commands correctly, translating them into precise movements and paths. This fidelity in animation not only ensures a true-to-life representation of the machining process but also provides an invaluable tool for planning and education, allowing users to visualize the end result before any physical machining takes place.

3 Design

In the design section of the CNC Simulation Program, we delve into the intricacies of a unique application blending Python's powerful backend capabilities with the dynamic visual strengths of HTML and JavaScript on the frontend. This application stands out as a tool for visualizing and simulating CNC machining operations, translating complex command sequences into understandable visual representations.

Figure 4: Technologies

3.1 Backend

The backend, built with Python, is the core where the processing of CNC commands takes place. It handles the interpretation of input commands, consisting of lines and arcs, and generates a structured data representation. This data includes detailed specifications of each command, like the coordinates for line segments and the parameters defining arcs. The Python script performs the crucial task of parsing these commands from user input, ensuring accuracy and consistency in the data fed to the frontend.

This server, set up within the Python script, plays a critical role in the application's functionality. It serves the frontend HTML and JavaScript files, enabling the visualization of the CNC simulations in a web browser. This integration of a web server signifies the application's versatility, allowing it to operate as a standalone tool with a web-based interface, thereby enhancing user accessibility and interactivity.

```python
def start_server():
    Handler = http.server.SimpleHTTPRequestHandler
    with socketserver.TCPServer(("", PORT), Handler) as httpd:
        print("Serving at port", PORT)
        httpd.serve_forever()
```

Listing 1: Server code

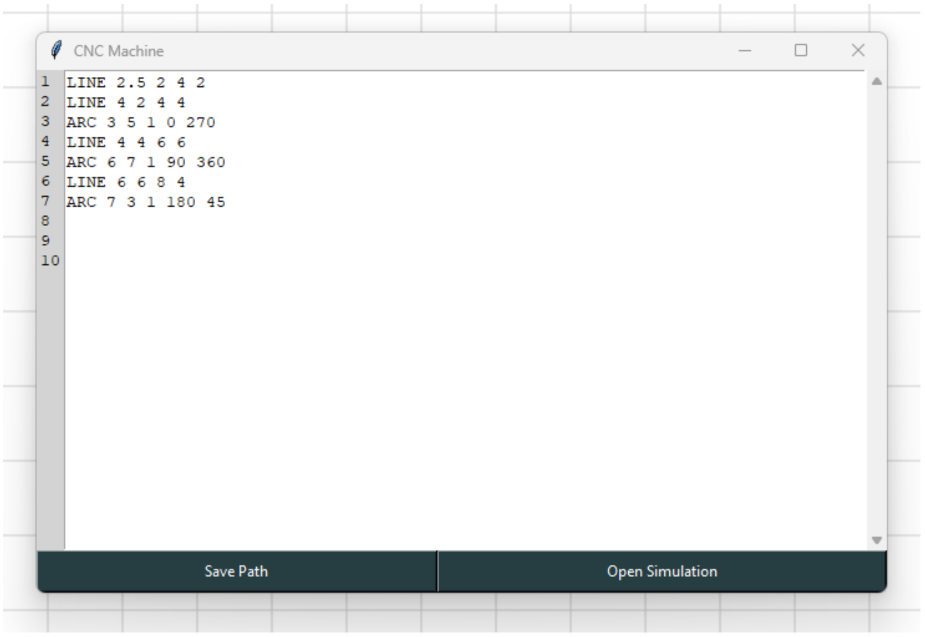
In the Python component of the CNC Simulation Program, alongside processing and interpreting CNC commands, there is a crucial functionality for handling operator inputs. Commands entered by the operator are stored in a text file. This text file is then parsed by the script, which meticulously converts each command into structured data. Subsequently, a new JSON file is generated, containing this structured data. This JSON file serves as a bridge to the frontend, where it is used to render the CNC paths visually. This process ensures a seamless flow of data from the operator's input to the final visual representation.

```python
def save_path_and_generate_commands(input_text):
    content = input_text.get("1.0", "end-1c")
    with open('path_file.txt', 'w') as file:
        file.write(content)

    filename = "path_file.txt"
    path = read_path_from_file(filename)
    commands = generate_commands(path)

    data = {
        "path": [
            {"type": "line", "start": [segment.start.x, segment.
    start.y], "end": [segment.end.x, segment.end.y]} if isinstance(
    segment, LineSegment) else
            {"type": "arc", "center": [segment.center.x, segment.
    center.y], "radius": segment.radius, "start_angle": segment.
    start_angle, "end_angle": segment.end_angle} for segment in
    path
        ],
        "commands": commands
    }

    with open('cnc_data.json', 'w') as f:
        json.dump(data, f, indent=4)

    print("Path saved and commands generated.")
```

Listing 2: Path code

3.2 Frontend

On the frontend, an HTML page with embedded JavaScript takes the baton, transforming the data into animated visualizations. It employs D3.js, a powerful library for data-driven visualizations, to render the command paths on a dynamically sized SVG canvas. This setup allows users to see the exact path that a CNC machine would follow based on the provided commands. The JavaScript logic also includes interactive elements and real-time adjustments, offering users an immersive experience in understanding and analyzing CNC machining processes.



```
LINE 2.5 2 4 2
LINE 4 2 4 4
ARC 3 5 1 0 270
LINE 4 4 6 6
ARC 6 7 1 90 360
LINE 6 6 8 4
ARC 7 3 1 180 45
```

Figure 5: Python menu

To enhance the frontend design of the CNC Simulation Program, we also incorporate a user interface created with Tkinter, a standard GUI toolkit in Python. This interface features a menu system that facilitates user interaction, allowing for the easy input and modification of CNC commands. The Tkinter-based menu not only simplifies the process of entering and editing the path commands for users but also integrates seamlessly with the Python backend, ensuring a cohesive and user-friendly experience. This combination of a graphical menu with the D3.js visualizations creates a comprehensive and interactive frontend for the program.

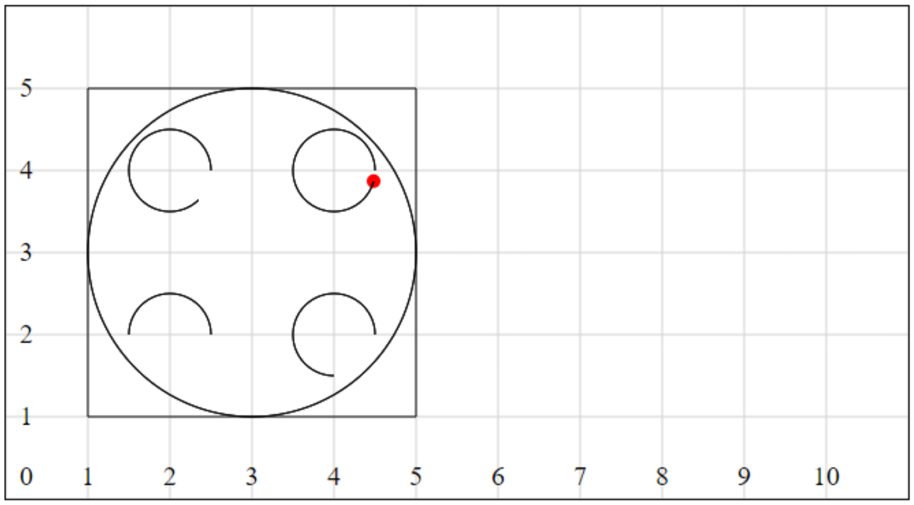


Figure 6: Example

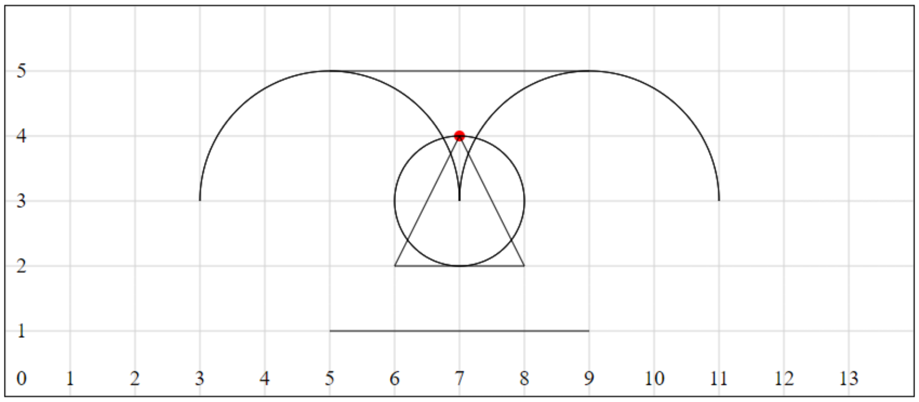


Figure 7: Example

4 Implementation

4.1 Animation and Drawing Functions

The core functionality of the frontend visualization lies in several JavaScript functions. The `animateSegment` function orchestrates the animation process, determining whether a line or arc needs to be drawn based on the segment type. It recursively calls itself to animate each segment in sequence.

```javascript
function animateSegment(path, index) {
            if (index >= path.length) return;

            const segment = path[index];
            if (segment.type === 'line') {
                moveCNCHead(segment.start, () => {
                    drawLine(segment, () => animateSegment(path, index + 1));
                });
            } else if (segment.type === 'arc') {
                const startAngleRadians = degreesToRadians(segment.start_angle);
                const coordinates = [
                    segment.center[0] + segment.radius * Math.cos(startAngleRadians),
                    segment.center[1] + segment.radius * Math.sin(startAngleRadians)
                ]
                moveCNCHead(coordinates, () => {
                    drawArc(segment, () => animateSegment(path, index + 1));
                });
            }
        }
```

Listing 3: Segment Animation Function

The `moveCNCHead` function handles the movement of the CNC head to the start of each new segment, providing a smooth transition between segments.

```javascript
function moveCNCHead(target, callback) {
    cncHead.transition()
        .duration(500)
        .attr("cx", target[0] * scaleFactor)
        .attr("cy", height - (target[1] * scaleFactor))
        .on("end", callback);
}
```

Listing 4: CNC Head Movement Function

4.2 Line Drawing Function

For line segments, `drawLine` is used, drawing a straight line from the start to the end point.

```
function drawLine(segment, callback) {
            const lineData = [
                [segment.start[0] * scaleFactor, height - (segment.start[1] * scaleFactor)],
                [segment.end[0] * scaleFactor, height - (segment.end[1] * scaleFactor)]
            ];

            svg.append("path")
                .datum(lineData)
                .attr("d", d3.line())
                .attr("stroke", "black")
                .attr("fill", "none")
                .attr("stroke-dasharray", function () {
                    const length = this.getTotalLength();
                    return `${length} ${length}`;
                })
                .attr("stroke-dashoffset", function () {
                    return this.getTotalLength();
                })
                .transition()
                .duration(500)
                .attr("stroke-dashoffset", 0)
                .on("end", callback);

            // Animate CNC head along the line
            svg.transition()
                .duration(500)
                .tween("pathTween", () => {
                    return t => {
                        const interpolateX = d3.interpolate(segment.start[0], segment.end[0]);
                        const interpolateY = d3.interpolate(segment.start[1], segment.end[1]);
                        cncHead
                            .attr("cx", interpolateX(t) * scaleFactor)
                            .attr("cy", height - (interpolateY(t) * scaleFactor));
                    };
                });
        }
```

Listing 5: Line Drawing Function

4.3 Arc Drawing Functions

The `drawArc` function in the JavaScript implementation plays a pivotal role in rendering arc segments. It calculates the start and end angles of each arc, using D3.js's arc generator to create the path. This path is then appended to the SVG canvas, representing the arc's trajectory.

```
function drawArc(segment, callback) {
        const startAngle = Math.PI / 2 - degreesToRadians(segment.
    start_angle);
        const endAngle = Math.PI / 2 - degreesToRadians(segment.
    end_angle);

        const arcGenerator = d3.arc()
            .innerRadius(segment.radius * scaleFactor)
            .outerRadius(segment.radius * scaleFactor)
            .startAngle(startAngle)
            .endAngle(endAngle);

        const arcPath = svg.append("path")
            .attr("d", arcGenerator)
            .attr("transform", `translate(${segment.center[0] *
    scaleFactor}, ${height - (segment.center[1] * scaleFactor)})`)
            .attr("fill", "none")
            .attr("stroke", "black")
            .attr("stroke-dasharray", function () {
                const length = this.getTotalLength();
                return `${length} ${length}`;
            })
            .attr("stroke-dashoffset", function () {
                return this.getTotalLength();
            });

        arcPath.transition()
            .duration(1000)
            .attr("stroke-dashoffset", 0)
            .on("end", callback);

        // Animate CNC head along the arc
        animateCNCHeadForArc(segment, 700);
}
```

Listing 6: Arc Drawing Function

In addition, the `animateCNCHeadForArc` function animates the movement of the CNC head along the arc, creating a smooth and continuous path animation. This visual representation is crucial for accurately depicting curved paths in the CNC machining process.

```javascript
function animateCNCHeadForArc(segment, duration) {
            const startAngle = degreesToRadians(segment.start_angle);
            const endAngle = degreesToRadians(segment.end_angle);

            svg.transition()
                .duration(duration)
                .tween("pathTween", () => {
                    return t => {
                        const interpolateAngle = d3.interpolate(startAngle, endAngle);
                        const angle = interpolateAngle(t);
                        const x = segment.center[0] + segment.radius * Math.cos(angle);
                        const y = segment.center[1] + segment.radius * Math.sin(angle);
                        cncHead
                            .attr("cx", x * scaleFactor)
                            .attr("cy", height - (y * scaleFactor));
                    };
                });
    }
```

Listing 7: CNC Head Animation for Arcs

These functions together create a dynamic and interactive visual representation of the CNC paths, crucial for understanding the machining process.

5 Conclusions

In conclusion, the CNC Simulation Program showcases a harmonious integration of backend processing and frontend visualization, reflecting a significant stride in CNC technology education and planning. The Python backend efficiently parses and structures CNC commands, while the frontend, utilizing D3.js, brings these commands to life through dynamic animations. This program not only enhances understanding and planning in CNC machining but also serves as an invaluable educational tool, bridging the gap between theoretical knowledge and practical application.

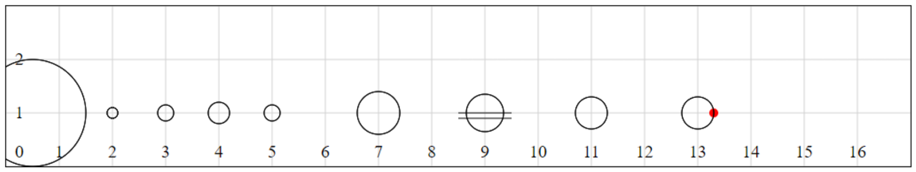


Figure 8: Solar System

6 Bibliography

1. Python Documentation. https://docs.python.org/3/
2. D3.js. Data-Driven Documents. https://d3js.org/
3. Wikipedia. https://wikipedia.com