GCode Animation with Python: G0-G1, G2-G3, G05*

Umut Özen

Department of Mathematics,
Dokuz Eylül University.

2024-2025 - Fall Semester

*Supervisor: Dr. Başak Karpuz

Abstract

Abstract

In this presentation, we introduce some GCodes for CNC (Computer Numerical Control) machines. Then, we introduce mathematical background related to motion and behavior GCodes of the CNC machine. Finally, we give our `Python` codes that creates an animation for a tangential cutter by reading a `.gcode` file.

Outline of the Talk

Below, we explain the procedure on how we retrieve information from weekly course schedules in `DEBIS`.

1. Examining the target web page on web browser
   1. Playing with the objects
   2. Inspecting the source
2. Reading data from the internet by using `Python`
   1. Getting list of academics
   2. Reading department information
   3. Getting weekly course data
3. Parsing data from the source by using `Python`
   1. Extracting timetable entries
4. Saving data to an `Excel Sheet` by using `Python`
5. Processing data in the excel sheet by using `Python`

GCodes

https://chatgpt.com/

GCodes

Some GCodes: Linear Motion G0-G1

...https://marlinfw.org/docs/gcode/G000-G001.html

GCodes

Some GCodes: Circular Motion G2-G3

. . . https://marlinfw.org/docs/gcode/G002-G003.html

GCodes

Some GCodes: Cubic Spline G5

. . . https://marlinfw.org/docs/gcode/G005.html

Curves Related to the GCodes: G0-G1, G2-G3, G5

Some GCodes: Linear Motion G0-G1

The line segment starting from the point P_0 and ending at the point P_1

$$\alpha(t) := P_0 + t(P_1 - P_0), \quad 0 \leq t \leq 1,$$

whose tangent slope is the constant value

$$\alpha'(t) := (P_1 - P_0), \quad 0 \leq t \leq 1,$$

Curves Related to the GCodes: G0-G1, G2-G3, G5

Some GCodes: Circular Motion G2-G3

The circular arc centered at the point P_0 with radius r, starting at an angle θ_0 and ending at an angle θ_1

$$\alpha(t) := P_0 + r\big(\cos(t), \sin(t)\big), \quad \theta_0 \leq t \leq \theta_1,$$

whose tangent slope is the circular curve

$$\alpha'(t) := r\big(-\sin(t), \cos(t)\big), \quad \theta_0 \leq t \leq \theta_1.$$

Curves Related to the GCodes: G0-G1, G2-G3, G5

Some GCodes: Cubic Spline G5

Cubic Bézier curve starting at the point P_0, ending at the point P_3 and with the additional control points P_1 and P_2

$$\alpha(t) := \sum_{i=0}^{3} \binom{3}{i} (1-t)^{3-i} t^i P_i, \quad 0 \le t \le 1,$$

whose tangent curve is

$$\begin{aligned} \alpha'(t) := & \sum_{i=0}^{2} \binom{3}{i} (3-i)(1-t)^{2-i} t^i P_i \\ & + \sum_{i=1}^{3} \binom{3}{i} i(1-t)^{3-i} t^{i-1} P_i, \quad 0 \le t \le 1. \end{aligned}$$

Animation Program Code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import comb
from scipy.integrate import quad
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
from scipy.spatial.transform import Rotation as R
from mpl_toolkits.mplot3d.art3d import Poly3DCollection


# G0/G1 [x,y,z,e,f,i,j,p,q]
# [[x1,y1,z1,e1,f1,p1,q1],[x2,y2,z2,e2,f2,p2,q2],...,[xn,yn,zn,en
    ,fn,pn,qn]]
def interpolate_points(x, y, z, steps=100):
    x_interp = np.linspace(x[0], x[1], steps)
    y_interp = np.linspace(y[0], y[1], steps)
    z_interp = np.linspace(z[0], z[1], steps)
    return x_interp, y_interp, z_interp
```

Animation Program Code

```
def cubic_bezier_derivative(t, control_points):
    # derivative_bernstein_coeffs = np.array([comb(3, i, exact=True) * (3-i) * (-1)**(3-i) * (1 - t) ** (2 - i) * t ** i for i in range(0,3)]+[comb(3, i, exact=True) * i * (1 - t) ** (3 - i) * t ** (i-1) for i in range(1,4)])
    # return np.dot(derivative_bernstein_coeffs, control_points[0:3]+control_points[1:4])
    return -3 * (1 - t) ** 2 * control_points[0] + (3 * (1 - t) ** 2 - 6 * (1 - t) * t) * control_points[1] + (
            6 * (1 - t) * t - 3 * t ** 2) * control_points[2] + 3 * t ** 2 * control_points[3]
```

```
def arc_length_integrand(t, control_points):
    derivative = cubic_bezier_derivative(t, control_points)
    return np.linalg.norm(derivative)
```


```
def arc_length_integrand(t, control_points):
    derivative = cubic_bezier_derivative(t, control_points)
    return np.linalg.norm(derivative)
```

Thank you very much for your interest to our talk.