# Emmanuel Eze, 20471601

# Software for a Drawing Robot

# Software Description

The drawing Robot software project is designed to generate and execute G-code commands for a drawing rig to reproduce input text given to it by a user. The program will allow the robot to perform this function by processing a file of font data and contained within is stroke definitions for each ASCII character.

**Font Data Processing**

* The strokes are read from a font file (SingleStrokeFont.txt) where each character is defined by a series of X, Y coordinates and a pen up/down state.

**Text Input**

* The user will be asked by the program to specify a file containing the text to be drawn. This file is subsequently read and processed by the program, therefore separating characters, offsets and line spacing.

**Scale Factor and offsetting**

* The required text height is set between 4-10mm. a scale factor is also employed to adapt the font’s predefined stroke dimensions accordingly. X and Y offsets manage the character and line spacing ensuring the generated G-codes align on the drawing surface.

**G-Code Generation**

* Each character extracted from the text file is translated into G-code commands. G0 (pen up) and G1 (pen down) indicate the drawing operations and robot’s maneuvers.

**Starting the Robot**

* The software includes functionality to wake the robot, initialise its drawing state, and transmit G-code commands. Communication between the robot hardware and the program is done through commands sent via a virtual RS232 interface.

**Application**

* The program operates in 2 modes. Simulation for outputting G-codes to the terminal and operational mode for sending G-codes directly to the robot to execute.

**Using Git for Version Control**

* Set up a version control repository using Git. Throughout development, make frequent commits to the original skeleton code. Make sure the repository has all generated code and documentation files committed.

**Limitations**

* Ensure the text doesn't break when it fits within a 100mm-wide writing space. Using the ASCII values for line feed (LF) and carriage return (CR), space consecutive lines 5 mm apart

# Project Files

(Maximum 1 page)

# Key Data Items

|  |  |  |
| --- | --- | --- |
| Name | Data type | Rationale |
| ASCII\_VALUE | int | Stores the ASCII code of the current character, to be looked up in the font data |
| Text\_Buffer | char[] | Stores the text input from the text file to be processed and converted into G-codes. |
| Font\_Data | Struct Character\_Font \* | Stores stroke details of the characters |
| gcode\_buffer | char[] | Temporarily holds the G-code commands before sending to robot |
| Current\_Ch\_Coord | struct StrokesCoord \* | Temporarily holds stroke data coordinates of character currently being processed |
| Text\_Height | int | Basis of determining scaling factor for all characters |
| TextPointer | FILE \* | Points to input text file from user for reading |
| StrokeFontPointer | FILE \* | Pointer to the font file, enabling reading of stroke definitions for characters |
| scale\_factor | float | Scales the font strokes based on the input text height |
| offset | int | Tracks the vertical and horizontal position for each character for proper positioning |
|  |  |  |

# Functions

bool Extract\_Single\_Stroke\_Data(FILE \*SingleStrokeFont)

* **SingleStrokeFont**- input file containing character stroke definitions.
* **Return** - returns true if file is successfully opened and false if not.

bool Extract\_Text\_String(FILE \*test, char \*Text\_Buffer, size\_t Buffer\_Size)

* **test**- input file containing text to be drawn
* **Text\_Buffer**- A buffer where the read text will be stored.
* **Buffer\_Size**- size of the buffer to ensure sufficient memory
* **Return**- returns true or false if file is successfully opened or not

void ASCII\_Character\_Search\_XYP\_Offset(char \*text, float Text\_Height, char \* gcode\_buffer)

* **test**- input file containing text to be drawn
* **Text\_Height**- input height between 4-10mm
* **gcode\_buffer**- buffer where generated G-Code are stored
* **Return**- no value returned

float Scale\_Factor(float Text\_Height)

* **Text\_Height**- input height between 4-10mm
* **Return**- no value returned

# Testing Information

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Test Case | Test Data | Expected Output |
| Extract\_Single\_Stroke\_Data(FILE \*SingleStrokeFont) | Font file with stroke data | SingleStrokeFont.txt | Loads stroke data into Font\_Data successfully |
|  | Font file invalid/empty | Invalid file | Message: "Error: Failed to load font data’’ |
| Extract\_Text\_String(FILE \*test, char \*Text\_Buffer, size\_t Buffer\_Size) | Text file with text input | test.txt with "The quick brown fox jumped over the lazy dog" | Text buffer contains "The quick brown fox jumped over the lazy dog". |
|  | Text file invalid/empty | Invalid file | Message: ‘’Could not open text file’’ |
| ASCII\_Character\_Search\_XYP\_Offset(char \*text, float Text\_Height, char \* gcode\_buffer) | Valid text height input | ‘‘Height = 8’’ | Generated G-Codes for each character correctly scaled for height of 8mm |
|  | Text input contains unrecognized characters | Non-ASCII characters in file | Message: ‘’ Warning: Unrecognised character’’ |
|  | Text input exceeds 100mm writing space | ‘’The quick brown fox jumped over the lazy dog’’ | G-Codes split lines with X\_offset = 0 and Y\_offset -= 5 |
| Scale\_Factor(float Text\_Height) | Valid height | E.g. ‘‘Height = 8mm’’ | Returns scale factor: 8 / 18 = 0.444. |
|  | Invalid height | Height < 4mm or Height > 10mm | Message: ‘’ Error: Text height must be between 4 and 10mm’’ |

# Flowchart(s)

May be included as separate pdf