# Mathis Bellino - 20342807

# **CNC Text Plotter System Manual**

# Software Description

The CNC Text Plotter software controls a robotic plotter to draw text using a single-stroke font system. The software reads characters’ stroke data from SingleStrokeFont.txt, processes it according to user-specified height requirements (4-10mm), and generates appropriate G-code commands to control the plotter's movements.

Key Features:

1. Font Processing
   * Dynamically allocates memory for font data
   * Scales character strokes based on desired height
   * Maintains proper character spacing
2. Plotting Text Layout
   * Uses X limit of 100mm and Y limit of 50mm
   * Maintains a minimum of 5mm line spacing + the user input height
   * Ensures complete words (no splitting across lines)
3. Robot Control
   * Generates real-time G-code commands
   * Manages pen up/down movements
   * Maintains synchronisation through acknowledgment system

Sample Emulator Output:

# Project Files

Core implementation files:

**main.c**

* Primary program control flow
* User interaction handling
* Configuration Required:
  + Verify COM port setting in serial.h
  + Check debugging flags if needed

**processTextFile**

* Processes input text files.
* Reads text file contents character by character and handles word-level and line-level operations.
* Integrates with other components for word processing, G-code generation, and font data utilisation.

**font.h**

* Core data structure definitions
* Function prototypes
* Error code definitions
* No configuration typically needed

**storeFontData.c**

* Font file parsing implementation
* Dynamic memory allocation
* Configuration:
  + Memory allocation limits can be adjusted
  + File path handling can be modified

**calculateWordWidth.c**

* Word dimension calculations
* Scaling operations
* No runtime configuration needed

**processWord.c**

* Word-level processing
* Position management
* Configuration:
  + Word spacing can be adjusted via constants

**generateGCode.c**

* G-code command generation
* Pen state management
* Configuration:
  + Feed rates can be modified
  + Command formatting can be adjusted

**handleLineBreak.c**

* Line wrapping logic
* Spacing management
* Configuration:
  + Line spacing can be modified (default 5mm)

**Configuration Notes:**

* COM port setting in serial.h (default: COM7)
* Simulator mode: Comment out #define Serial\_Mode
* Robot mode : Uncomment #define Serial\_Mode

# Key Data Items

|  |  |  |
| --- | --- | --- |
| Name | Data type | Rationale |
| MovementData | **struct** | Contains x and y coordinates as floats for precision in stroke positions, and penDown as integer since pen state is binary (up/down). Structure grouping ensures data coherency and matches the font file format specification. |
| CharacterData | **struct** | Groups stroke count (int) with a dynamic array of MovementData. Dynamic allocation allows memory-efficient storage of characters with varying numbers of strokes, while the count enables proper iteration. |
| |  | | --- | | FontData |  |  | | --- | |  | | **struct** | Main container with fixed 256-element array of CharacterData. Size chosen to support full ASCII range while enabling direct character lookup through ASCII value indexing. A fixed array is preferred over a dynamic one since ASCII range is constant. |
| |  | | --- | | currentY  currentX |  |  | | --- | |  | | **float\*** | Pointers to current drawing position coordinates. Float type necessary for sub-millimetre precision in scaling and positioning. Pointer type allows multiple functions to update position values. |
| buffer | **char[100]** | Character array for constructing G-code commands. Size accommodates longest possible command string including coordinates, feed rate, and null terminator while maintaining safety margin. |
| font | **struct FontData\*** | Pointer to dynamically allocated font data structure. Pointer type chosen to enable efficient memory management of large font dataset and convenient passing between functions. |
| word | **char[256]** | Buffer for storing individual words during text processing. Size accommodates longest reasonable word plus null terminator. Fixed size chosen for simplicity. |
| height | **float** | Stores user-specified text height in millimeters (4.0-10.0mm range). Float type required for sub-millimeter precision in scaling calculations. |
| scaleFactor | **float** | Stores height/18.0 ratio used for coordinate scaling. Float type essential for precise conversion between font units and millimeters. |

# Functions

**storeFontData** - Loads and parses font definition file into memory using dynamic allocation.

***struct FontData\* storeFontData(const char\* filename)***

Description: Processes SingleStrokeFont.txt to create in-memory font data structures. Parameters:

* filename (input) -- Path to font definition file
* Return value -- Pointer to allocated FontData or NULL on error Error Cases:
* File not found: Returns NULL
* Memory allocation failure: Returns NULL
* Invalid file format: Returns NULL

Usage Notes:

- Must be called once at program start

- Returned data must be freed with freeFontData() when done

- Handles memory allocation automatically

**calculateWordWidth -** Calculates the total width a word will occupy when drawn at specified height.

**float calculateWordWidth(const char\* word, struct FontData\* font, float height)**

Parameters:

word – input string to calculate width for

font - input pointer to font data structure

height - input desired text height in millimeters

Return value - returns calculated width in millimeters, 0.0f if parameters invalid

**processWord:**

Processes a single word for plotting, handling positioning and line breaks.

Manages character drawing and spacing control.

**int processWord(const char\* word, struct FontData\* font, float\* currentX, float\* currentY, float height)**

Parameters:

word - input word to process

font - input pointer to font data

currentX - input/output pointer to current X position

currentY - input/output pointer to current Y position

height - input tex height for scaling

Return value - returns SUCCESS (0) or appropriate error code

**generateGCodeForCharacter**

Converts character stroke data into G-code commands for the plotter.

Creates pen movements and coordinate scaling.

**int generateGCodeForCharacter(char c, struct FontData\* font, float\* currentX, float\* baselineY, float height)**

Parameters:

Char c - input character to generate code for

font - input pointer to font data

currentX - input/output pointer to current X position

baselineY - input/output pointer to baseline Y position

height - input text height

Return value - returns SUCCESS (0) or error code

**processTextFile** - Primary text processing function that handles file reading and word generation. Controls overall text layout and drawing process by passing data to processWord.  
**int processTextFile(const char\* filename, struct FontData\* font, float height)**   
Parameters:

* filename - input path to text file to process
* font - input pointer to font data structure
* height - input text height for drawing
* Return value - returns SUCCESS (0) or appropriate error code

**handleLineBreak**

Manages text wrapping and line spacing calculations. Updates positioning for new lines.

**int handleLineBreak(float\* currentX, float\* currentY, float\* height)**

Parameters:

currentX - output pointer to X position (reset to 0)

currentY - input/output pointer to Y position for next line

height - input text height for spacing calculation

Return value - returns SUCCESS (0) or ERROR\_OUT\_OF\_BOUNDS if exceeds page

# Testing Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Test Case** | **Test Data** | **Expected Output** |
| main | Valid height input | "6.0" | Success, program continues |
|  | Invalid height (low) | "2.0" | Error message, retry prompt |
|  | Invalid height (high) | "12.0" | Error message, retry prompt |
|  | Invalid input type | "abc" | Error message, retry prompt |
|  | Valid text file | "input.txt" | Success, process text |
|  | Invalid text file | "missing.txt" | Error message, exit |
| storeFontData | Valid font | "SingleStrokeFont.txt" | FontData\* returned |
|  | Invalid file | "nonexistent.txt" | NULL, error logged |
|  | Empty file | "empty.txt" | NULL, error logged |
|  | Memory error | [Large dataset] | NULL, error logged |
|  | Corrupted file | "corrupt.txt" | NULL, error logged |
| calculateWordWidth | Standard word | "Test", h=6.0 | ~24.0mm |
|  | Empty string | "", h=6.0 | 0.0, error code |
|  | Scale error | "Test", h=2.0 | 0.0, error code |
|  | Max width test | "WWWW", h=10.0 | ~99mm |
|  | NULL parameters | NULL, font, 6.0 | 0.0, error code |
| generateGCodeForCharacter | Basic char | 'A', h=6.0 | G-codes generated |
|  | Invalid char | '#', h=6.0 | ERROR\_NO\_STROKE |
|  | Scale error | 'A', h=11.0 | ERROR\_INVALID\_PARAM |
|  | Null pointer | NULL font ptr | ERROR\_INVALID\_PARAM |
|  | Zero height | 'A', h=0.0 | ERROR\_INVALID\_PARAM |
| processWord | Simple word | "Hello", x=0, y=0 | SUCCESS |
|  | Word at line end | "Test", x=90, y=0 | Line break triggered |
|  | Empty word | "", x=0, y=0 | ERROR code |
|  | NULL parameters | NULL, font, ptrs | ERROR\_INVALID\_PARAM |
|  | Invalid height | "Test", h=11.0 | ERROR\_INVALID\_PARAM |
| handleLineBreak | Standard break | x=90,y=-6,h=6 | x=0,y=-12 |
|  | Page limit | x=0,y=-45,h=6 | ERROR\_OUT\_OF\_BOUNDS |
|  | Invalid height | x=0,y=0,h=11 | ERROR\_INVALID\_PARAM |
|  | NULL parameters | NULL ptrs | ERROR\_INVALID\_PARAM |
| processTextFile | Valid file | "test.txt", h=6 | SUCCESS |
|  | Empty file | "empty.txt", h=6 | SUCCESS |
|  | Missing file | "missing.txt", h=6 | ERROR code |
|  | Format error | "binary.dat", h=6 | ERROR code |
|  | Memory error | [Large file] | ERROR code |
| freeFontData | Valid structure | Populated FontData\* | Memory freed |
|  | NULL pointer | NULL | No action, returns |
|  | Empty structure | FontData\* no chars | Memory freed |
|  | Partial structure | Incomplete FontData\* | Partial memory freed |

# Flowchart(s)

May be included as separate pdf