Name & Student ID

Name: Tesimimode Megbele

Student ID: 20464073

Robot Writing Software Project

Software Description

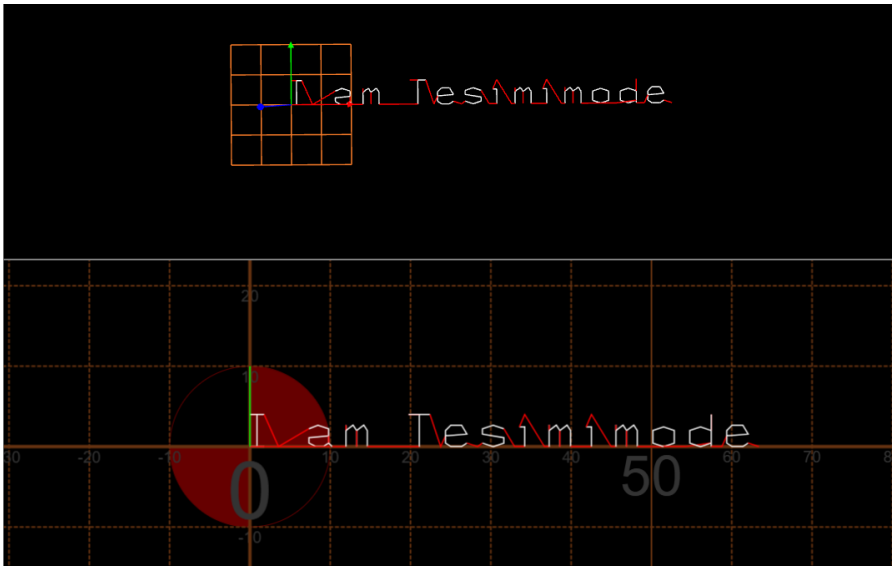
This software converts plain text files into G-code instructions for a CNC robot to write text on paper using a pen. The program reads words from "InputText.txt", looks up single-stroke font data from "SingleStrokeFont.txt", scales characters to a user-specified height (4-10mm), positions them within a 100mm×50mm drawing area with 5mm line spacing, and transmits G-code via serial port to an Arduino which controls the robot.

Software Workflow:

1. Reads words sequentially from input file
2. Matches each character's ASCII codes to stroke coordinates found in the 'SingleStrokeFont.txt' file, which is setup to give X, Y and Z position coordinates on separate lines.
3. Scales font units to physical mm, based on user font size input, applies the suitable character, word and line spacing and, carries out text wrapping whenever necessary.
4. Sends these processed coordinates as G-code, which is a specific format for the writing instructions to be interpreted by the robot correctly.
5. Sends commands at 115200 baud via COM port.

The software applies standard G-code syntax with M3 (pen enable), F1000 (feed rate) and the pen positioning coordinates. It also ensures that the pen always ends in UP position (S0). The G-code commands can be printed, therefore enabling compatibility with a web G-code simulator for testing sample text files.

An example output from the software is shown below, where the input text was "I am Tesimimode", with a font size of 4.



Project Files

File Name	Purpose	Key Functions
main.c	Facilitates the whole software, providing main processing loop for word processing and sending commands.	All functions
TexttoWordArray.c	Turns input text file into individual words, stored into an array.	TexttoWordArray()
WordArraytoASCII.c	Converts word characters to ASCII decimal numbers.	WordArraytoASCII()
ExtractStrokeData.c	Loads stroke coordinates from font file, based on ASCII numbers.	ExtractStrokeData(), LoadStrokeForChar()
ScaleandAdjustStrokeData.c	Scales and positions strokes within 100×50mm boundary.	ScaleandAdjustStrokeData()
ConvertStrokestoGcode.c	Generates G-code formatting for robot.	ConvertStrokestoGcode()
FreeStrokeData.c	Dynamic memory cleanup for stroke arrays.	FreeStrokeData()
InputText.txt	User text to draw (e.g., "hello world").	User input for text
SingleStrokeFont.txt	Stroke font database (999 ASCII nMoves + coordinates).	Lab font data
rs232.h, serial.h	Serial communication library.	PrintBuffer(), WaitForDollar()

Key Data Items

Name	Data Type	Rationale
StrokeData	Structure	Holds complete stroke information for one character: ASCII code, stroke count, X, Y and Z coordinate arrays. Enables the character processing.
FontSize	float	User-specified font height in mm (4-10 range). Used for scaling factor calculation. Float precision prevents rounding/ calculation errors.
TextToAscii	int[64]	Array storing ASCII decimal codes for characters in current word. Fixed size 64 accommodates a typical word length + safety margin in case of a very large word.
word	char[64]	Temporary buffer holding one word read from input file. Sized to match TextToAscii.
buffer	char[100]	Reusable character array for sprintf G-code formatting and serial transmission. Sized for longest command + safety margin.
curX, curY	float	Current drawing cursor position (mm). curX tracks horizontal position, curY tracks vertical baseline. Float to accommodate coordinates with decimals values.
chars	StrokeData[64]	Array holding stroke data for all characters in current word. Fixed size matches word buffer capacity.
word_count	int	Counter tracking total words successfully processed and drawn. Used for final summary output.

Functions

TexttoWordArray(FILE *file, char *word_buffer, int maxLengthWord)

Reads one complete word from the 'InputText.txt' text file into buffer. The word is extracted and stored in a string array.

Parameters: file (open text file), word_buffer, maxLengthWord

Return: 1 (word successfully read), 0 (end of file/ invalid input)

WordArraytoASCII(const char *word, int *TextToAscii, int maxLengthASCII)

Converts null-terminated string to array of ASCII decimal values.

Parameters: word (source string), TextToAscii (destination array), maxLengthASCII (max output length)

Return: Number of characters converted

ExtractStrokeData(const int *TextToAscii, int len, FILE *stroke_data, StrokeData *chars, int maxChars)

Searches font file for each ASCII decimal then loads the corresponding stroke coordinates into chars array.

Parameters: TextToAscii (input codes), len (character count), stroke_data (open font file), chars (output array), maxChars (max array size)

Return: Number of characters loaded, -1 (missing stroke data/ array too small)

ScaleandAdjustStrokeData(StrokeData *chars, int nChars, float FontSize, float *curX, float *curY, float maxWidth, float maxHeight)

Scales stroke coordinates to correct size and positions and, applies character spacing, all within the defined bounds and with text wrapping when needed.

Parameters: chars (stroke data), nChars (count), FontSize (mm), curX/curY (cursor pointers), maxWidth/Height (100×50mm bounds)

Return: void (updates arrays and cursor position pointers)

ConvertStrokestoGcode(StrokeData *chars, int nChars, char *buffer)

Generates G-code from the correctly positioned stroke coordinates. Applies the correct pen states to make sure pen is down or up when appropriate.

Parameters: chars (scaled strokes), nChars (count), buffer (G-code formatting)

Return: void (commands transferred to buffer)

FreeStrokeData(StrokeData *stroke)

Safely deallocates X, Y and Z arrays, resets pointers to NULL, preventing overuse of memory.

Parameters: stroke (single character data structure)

Return: void (stroke data structure reset to empty state)

Testing Information

Function	Test Case	Test Data	Expected Output
main()	Short text file	InputText.txt: "the quick brown fox" FontSize=6	Draws 4 words, wraps correctly, ends with S0, prints "Drew 4 words | X= Y="
main()	Long text exceeding bounds	"hello world..." (60+ chars) FontSize=8	Draws until Y < -50mm, skips remainder, no crash
main()	Invalid files	Delete InputText.txt	"Could not open InputText.txt" with Exit code 1
main()	Invalid FontSize	Enter "-5" or "999"	Forces size to 4.0 or 10.0, continues normally
TexttoWordArray()	Normal word	File: "hello world"	Returns "hello" (buffer), return= 1
TexttoWordArray()	Multiple whitespace	File: " hello world "	Returns "hello" then "world", skips all spaces
TexttoWordArray()	End of File (EOF)	File empty	Returns 0, buffer= '\0'
TexttoWordArray()	Long word	70-char word, maxLengthWord= 64	Copies first 63 chars + '\0', truncates word
WordArraytoASCII()	Normal word	"hello"	TextToAscii=, return= 5
WordArraytoASCII()	Buffer limit	70-char word, maxLengthASCII= 10	Copies first 10 chars, returns 10
ExtractStrokeData()	Valid ASCII range	TextToAscii= ('hello')	Loads 5 StrokeData structs, return= 5
ExtractStrokeData()	Missing character	TextToAscii= (no strokes)	Returns -1 (missing stroke data)
ScaleandAdjustStrokeData()	Normal word	"hello", FontSize= 6, curX= 0	Scales to 6mm height, positions at (0,0), updates curX
ScaleandAdjustStrokeData()	Word wrapping	Long word at curX= 90, FontSize= 8	Resets position to curX= 0, curY = -13 (8 font height + 5 line spacing)
ScaleandAdjustStrokeData()	Vertical overflow	curY= -55, maxHeight= 50	Returns immediately (no drawing)

ConvertStrokestoGcode()	Single stroke	1 char with 3 strokes (up/down/up)	Sends: S1000, G1, S0, G0
ConvertStrokestoGcode()	Pen state checking	Multiple pen changes	Only sends S0/S1000 when state actually changes
FreeStrokeData()	Normal case	StrokeData with 10 allocated strokes	All 3 arrays freed, pointers= NULL, nMoves= 0
FreeStrokeData()	NULL input	stroke= NULL	Returns immediately (no crash)
FreeStrokeData()	Empty strokes	nMoves= 0	Returns correctly (no memory to free)

AI Statement

No artificial intelligence tools were used in the development of the software code. AI was exclusively employed to enhance my understanding of C programming syntax, memory management principles and serial communication. This additional knowledge enabled more efficient implementation of coding architecture, helping to save time spent actually coding the software.

Flowchart(s)

Flowcharts included in the zip file under file names starting with: SystemFlowchartTM_20464073.